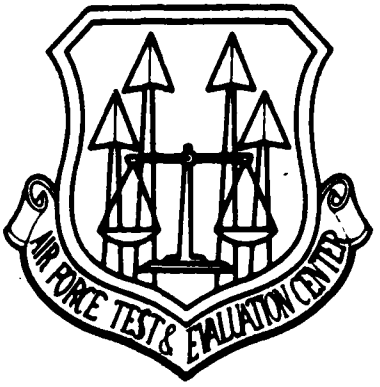
LEVEL II

12

AD A104328



SOFTWARE OT&E GUIDELINES

VOLUME III

SOFTWARE MAINTAINABILITY EVALUATOR'S HANDBOOK

APRIL 1980

AIR FORCE TEST AND EVALUATION CENTER
KIRTLAND AIR FORCE BASE
NEW MEXICO 87117

AFTECP 800-3

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFTECP-800-3	2. GOVT ACCESSION NO. AD-A104 328	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE OT&E GUIDELINES, Volume III (of five), "Software Maintainability Evaluator's Handbook"		5. TYPE OF REPORT & PERIOD COVERED Final, April 1980 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) N/A		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS HQ Air Force Test and Evaluation Center AFTEC/TEBC Kirtland AFB NM 87117		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS N/A
11. CONTROLLING OFFICE NAME AND ADDRESS Same as 9.		12. REPORT DATE Apr 80 13. NUMBER OF PAGES 233
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

AFTEC/TEBC point of contact:
Lt Col Michael A. Blackledge, USAF

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software
Software Testing
Software Evaluation
Quality Factors
Software Maintenance
Operational Test & Evaluation (OT&E)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Software OT&E Guidelines is a set of handbooks prepared by the Computer/Support Systems Division of the Test and Evaluation Directorate of the Air Force Test and Evaluation Center (AFTEC) for use in the operational test and evaluation (OT&E) of software. Volumes in the set include: 1) Software Test Manager's Handbook (AFTECP 800-1); 2) Handbook for the Deputy for Software Evaluation (AFTECP 800-2); 3) Software Maintainability Evaluator's Handbook (AFTECP 800-3); 4) Software Operator-Machine Interface Evaluator's Handbook (continued)

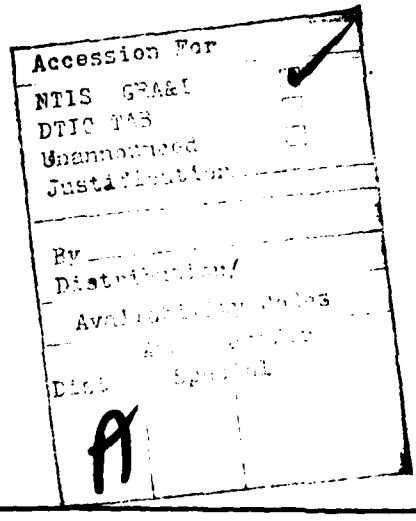
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 continued.

(AFTECP 800-4); Software Support Facility Evaluation Tools User's Handbook (AFTECP 800-5). The Software Maintainability Evaluator's Handbook was prepared as a guide for the software evaluator participating in AFTEC's software maintainability evaluation process. It provides a description of AFTEC's methodology for performing this evaluation, as well as guidelines by which members of a team of evaluators can independently evaluate a body of software as to its maintainability. The handbook includes the characteristics by which both the source code and documentation are rated, as well as examples, explanations, and definitions.



SOFTWARE OT&E GUIDELINES

VOLUME III

SOFTWARE MAINTAINABILITY

EVALUATOR'S HANDBOOK

APRIL 1980

AIR FORCE TEST AND EVALUATION CENTER
KIRTLAND AFB, NEW MEXICO 87117

FOREWORD

This volume is one of a set of handbooks prepared by the Computer/Support Systems Division of the Test and Evaluation Directorate, Air Force Test and Evaluation Center (AFTEC) for use in the operational test and evaluation of software. Comments should be directed to AFTEC/TEB, Kirtland AFB, NM 87117. Volumes in the set include:

I. Software Test Manager's Handbook (AFTECP 800-1).

II. Handbook for the Deputy for Software Evaluation (AFTECP 800-2).

III. Software Maintainability Evaluator's Handbook (AFTECP 800-3).

IV. Software Operator-Machine Interface Evaluator's Handbook (AFTECP 800-4).

V. Software Support Facility Evaluation Tools User's Handbook (AFTECP 800-5).

SOFTWARE MAINTAINABILITY
EVALUATOR'S HANDBOOK

Table of Contents

ILLUSTRATIONS

PART I

SOFTWARE MAINTAINABILITY EVALUATOR'S HANDBOOK
EVALUATOR GUIDELINES

A. GENERAL.

1. Purpose.

The purpose of this handbook is to provide to the software evaluator the information needed to participate in the Air Force Test and Evaluation Center's (AFTEC's) software maintainability evaluation process. Software maintainability is determined by those characteristics of software and computer support resources which affect the ability of software programmer/analysts to change software. Such changes are made to:

a. Correct errors.

b. Add system capabilities.

c. Delete features from programs.

d. Modify software to be compatible with hardware changes.

In this handbook, "software maintainability" is limited to software design and documentation assessments.

2. Use.

This handbook is divided into three parts.

a. The first part provides the evaluator with: (1) a background of the AFTEC software maintainability evaluation concept, (2) a basic understanding of the evaluation procedures, and (3) detailed instructions for using AFTEC's standard software maintainability questionnaires and answer sheets. The evaluator should read part I in its entirety

and understand the evaluation concept and procedures prior to beginning any evaluation.

b. Part II contains the questionnaires and explanatory information on each question. This information is provided in an attempt to ensure the evaluator fully understands the intent of each question. Included are definitions of terms, examples, explanations, and special case response instructions, as necessary. Each evaluator is encouraged to make notes in his copy of the handbook which will help clarify the intent of the questions. Part II of this handbook is designed to be used as the source of questions for the evaluation.

c. The final part of the handbook is a cross reference index. Once an evaluator has confidence in his understanding of the intent of each question, a short form of the questionnaire may be used. The cross reference index will then allow the evaluator to quickly locate any needed information in the handbook.

B. MAINTAINABILITY EVALUATION METHODOLOGY.

The methodology for evaluating software maintainability is based on the use of closed form questionnaires with optional written comments. These questionnaires are designed to determine the presence or absence of certain desirable attributes in a given software product. The elements of software maintainability and their relationships are shown in Figure 1 and described in following paragraphs. The hierarchical evaluation structure shown in the figure enables us to identify potential maintainability problems at various levels: category (documentation, source listings), characteristic (modularity, consistency, etc.), or a combination of the two.

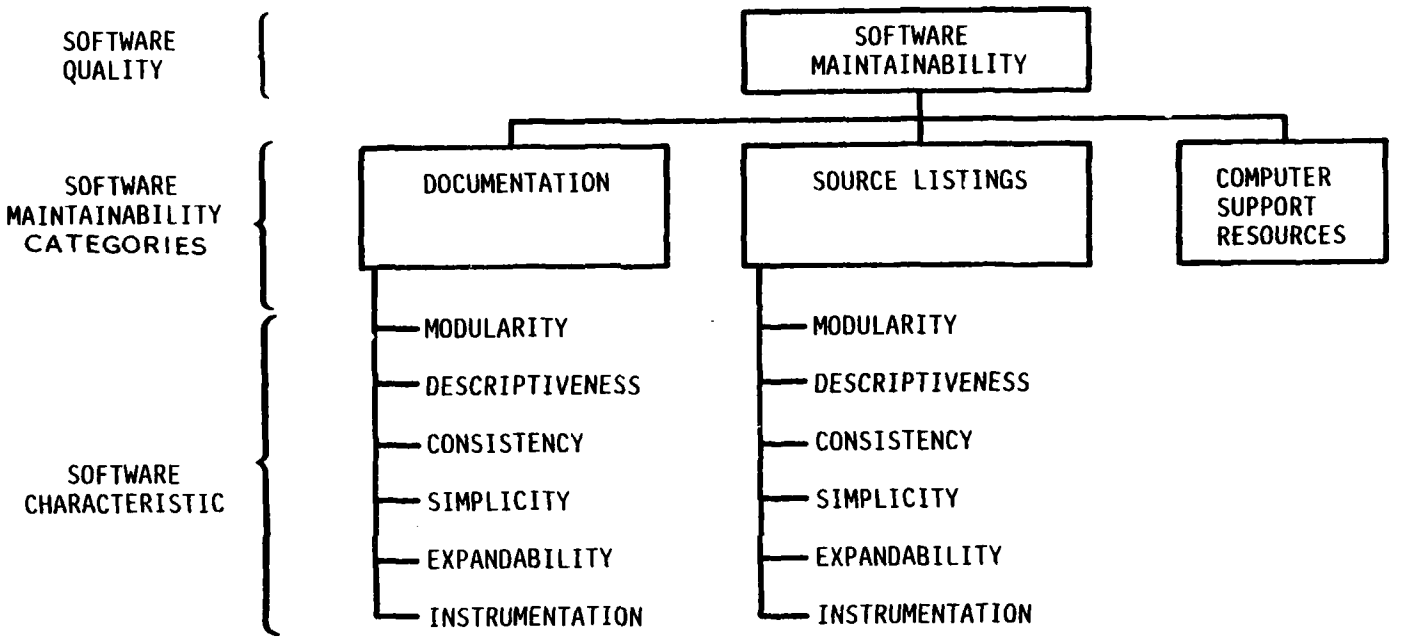


Figure 1. Elements of software maintainability

1. Software Categories.

Software consists of a set of computer instructions and data structured into programs, and the associated documentation on the design, implementation, test, support, and operation of those programs. Each software program is separately evaluated and consists of a set of components called modules. A module may, in general, be at any conceptual level of the program. In FORTRAN, modules are generally defined to be subroutines; in COBOL a module is usually a program. The software test manager must decide on the definition of module for the specific language and system to be evaluated. For each program there are related categories which are evaluated for characteristics which affect its maintainability. The categories, or products, are the software documentation, the software source listings, and the computer support resources. It is important to emphasize that only products that will be available to the maintenance programmer are to be considered in an evaluation.

a. Software Documentation.

Software program documentation is the set of requirements, design specifications, guidelines, operational procedures, test information, problem reports, etc. which in total form is the written description of a computer program. The primary documentation which is used in this evaluation consists of the documents containing program design specifications, program testing information and procedures, and program maintenance information. These documents may have a variety of physical organizations depending upon the particular application. The documents are evaluated both for content and for general physical structure (format). The content evaluation is primarily concerned with how well the overall program has been designed (as documented) for maintainability. The format evaluation is primarily aimed at how the physical structure of the documentation (table of contents, index, numbering schemes, modular separation of parts, etc.) aids in understanding or locating program information.

b. Software Source Listings.

Software source listings are the computer generated (or equivalent) form of the program code in its source language (e.g, FORTRAN, COBOL, JOVIAL, Ada, assembly language, etc.). The source listing represents the program as implemented, in contrast to the documentation which for the most part represents the program design or implementation plan. In essence, source listings are also a form of program documentation, but for this maintainability evaluation, a distinction is made.

The source listing evaluation consists of a separate evaluation of each selected module's source listing and the consistency between the module's source listing and the related module documentation. The separate module evaluations are summarized to yield an overall evaluation of the software source listing for the given program.

c. Computer Support Resources.

Computer support resources include all the relevant resources such as software, computer equipment, facilities etc., which will be used to support the maintenance of the software being evaluated. Characteristics of and procedures for the evaluation of computer support resources will be detailed in a separate document.

2. Software Maintainability Test Factors.

The maintainability of software documentation and source listings is determined by examining six characteristics or test factors: modularity, descriptiveness, consistency, simplicity, expandability, and instrumentation. Definitions of these test factors and discussions of their application in the evaluation of the documentation and source listings are given in the following paragraphs.

a. Modularity.

Software possesses the characteristic of modularity to the extent a logical partitioning of software into parts, components, and/or modules has occurred.

Software that is the easiest to understand and change is composed of independent modules. Each software product is therefore evaluated in relation to the extent to which its logical parts, components, and modules are independent. The fewer and simpler the connections between parts, the easier it is to understand each module without reference to other parts. Minimizing connections between parts also minimizes the paths along which changes and errors can propagate into other parts of the system, thus reducing the occurrence of side-effects within the system.

As a general guideline, modularity implies that a given module consists of only a few easily recognizable functions which are closely related and that a minimal number of links exist to other modules - preferably only via parameters passed in a calling parameter list. In addition, the physical format of the documentation should exhibit component independence for its sections, volumes, etc. There should be separate sections for the description of the major parts which a given document's purpose encompasses.

b. Descriptiveness.

Software possesses the characteristic of descriptiveness to the extent that it contains information regarding its objectives, assumptions, inputs, processing, outputs, components, revision status, etc.

This attribute is very important in understanding software. Documentation should have a descriptive format and contain useful explanations of the software program design. The objectives, assumptions, inputs, etc., are useful (in varying

degrees of detail) in both documentation and source listings. In addition, the descriptiveness of the source language syntax and the judicious use of source commentary greatly aids efforts to understand the program operation.

c. Consistency.

Software possesses the characteristic of consistency to the extent the software products correlate and contain uniform notation, terminology and symbology.

The use of standards in documentation, flow chart construction and certain conventions in I/O processing, error processing, module interfacing, naming of modules/variables, etc. are typical reflections of consistency. Attention to consistency characteristics can greatly aid in understanding the program. Consistency allows one to generalize easily. For example, programs using consistent conventions require that the format of modules be similar. Thus by learning the format of one module (preface block, declaration format, error checks, etc.) the format of all modules is learned. This allows one to concentrate on understanding the true complexities of an algorithm, data structure, etc.

d. Simplicity.

Software possesses the characteristic of simplicity to the extent that it lacks complexity in organization, language, and implementation techniques and reflects the use of singularity concepts and fundamental structures.

The aspects of software complexity (or lack of simplicity) that are emphasized in the evaluation relate primarily to the concepts of size and primitives. The less there is to discriminate and the more use there is of basic or primitive techniques, structures, etc. the simpler the software will tend to be.

The use of a high order language as opposed to an assembly language tends to make a program simpler to understand because there are fewer discriminations which have to be made. There are certain programming considerations such as dynamic allocation of resources and recursive/reentrant coding which can greatly complicate the data and control flow. Real-time programs, because of the requirement for timing constraints and efficiency, tend to have more control complexity. The sheer bulk of a module (number of operators, operands, nested control structures, nested data structures, executable statements, statement labels, decision parameters, etc.) will determine to a great extent how simple or complex the source code is. While it is recognized that the particular application itself may preclude the possibility of a reasonably simple design or implementation because of requirements such as a particularly complex real-time scheduling algorithm or high level mathematical or other theoretical considerations, this complexity nonetheless makes maintenance more difficult.

e. Expandability.

Software possesses the characteristic of expandability to the extent that a physical change to information, computational functions, data storage or execution time can be easily accomplished once the nature of what is to be changed is understood.

Software may be perfectly understandable but not easily expandable. If the design of the program has not allowed for a flexible timing scheme or a reasonable storage margin, then even minor changes may be extremely difficult to implement. Parameterization of constants and basic data structure sizes usually improves expandability. It is also very important that the documentation include explanations of how to effect increases/decreases in data structure sizes or changes to the timing scheme, and the limitations of such program expandability should be clear.

The numbering schemes for source listings, documentation narrative, and graphic materials must be carefully considered so that physical modifications to the code and documentation can be easily accomplished when necessary.

f. Instrumentation.

Software possesses the characteristic of instrumentation to the extent it contains aids which enhance testing.

For the most part the documentation is evaluated on how well the program has been designed to include test aids (instruments), while the source listings are evaluated on how well the code seems to be implemented to allow for testing through the use of such test aids. This part of the evaluation reflects the concern (from a maintainability viewpoint) that the software be designed and implemented so that instrumentation is either imbedded within the program, can be easily inserted into the program, or is available through a support software system, or is available through a combination of these capabilities.

3. Software Maintainability Evaluation Procedure.

The basic software evaluation procedure involves four distinct phases: planning, calibration, assessment, and analysis, as shown in Figure 2.

During the planning phase, the software test manager and the deputy for software evaluation (DSE) establish an evaluator team consisting of at least five evaluators knowledgeable in software maintenance. The program/module hierarchy is established and a set of representative modules is selected for each program to be evaluated. This set of modules is chosen by the DSE, as advised by the software test manager. The schedule for the evaluation is also established at this time. The software test manager briefs the evaluator team on the procedures and assigns the necessary identification information for this specific evaluation.

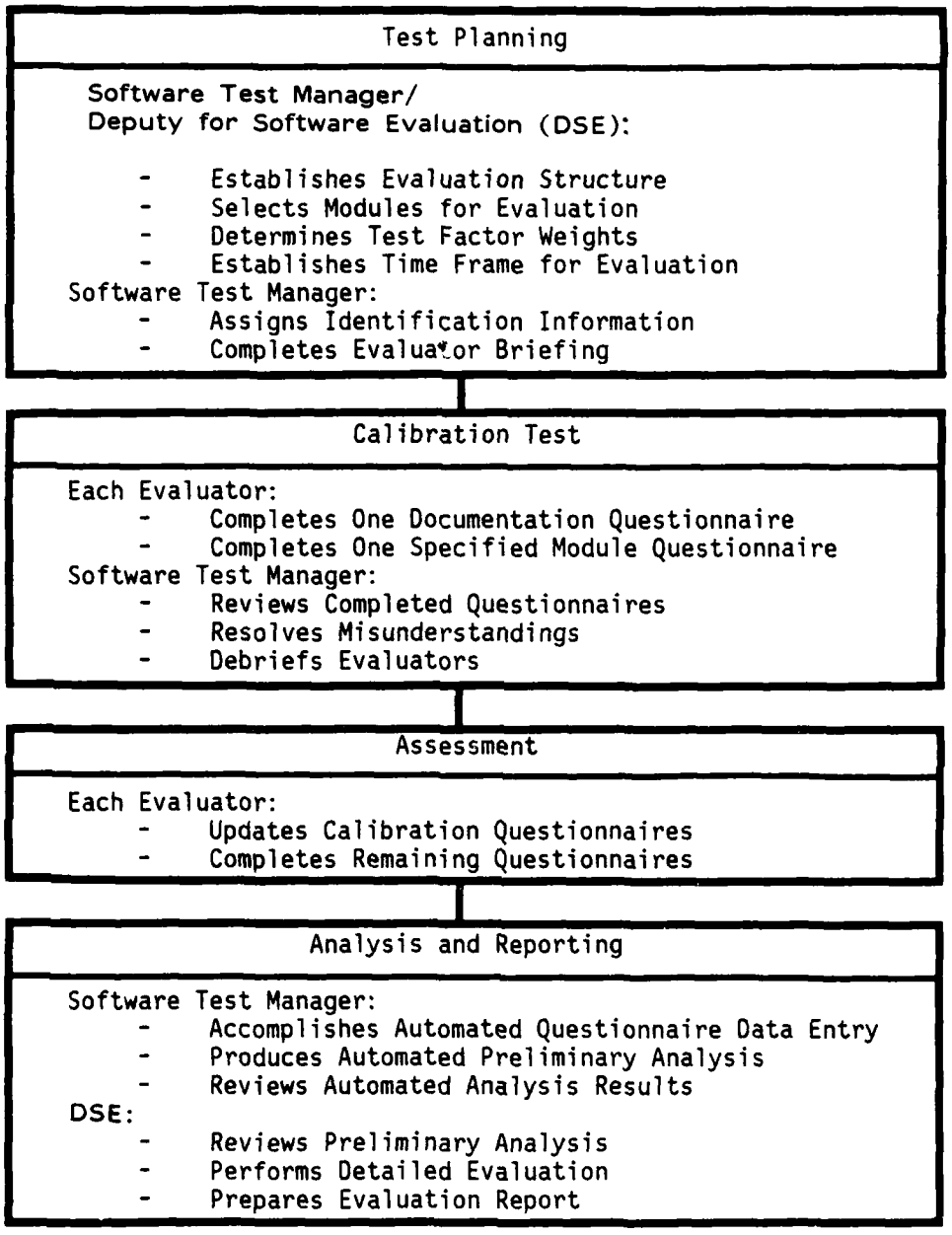


Figure 2. Maintainability Evaluation Procedure

The function of the calibration phase is to assure a reliable evaluation by ensuring each evaluator has a clear understanding of the questions on each questionnaire and their specific response guidelines. Each evaluator completes a documentation and a module source listing questionnaire in a trial or calibration evaluation. The completed questionnaires are reviewed to detect areas of misunderstanding and the evaluation teams are debriefed on the problem areas.

In the assessment phase, the evaluation teams update their calibration test questionnaires based on the results of the calibration debriefing. The teams then complete the remainder of their assigned documentation and module source listing questionnaires.

In the analysis phase, the software test manager accomplishes the conversion and initial data processing of the questionnaire data. The statistical summaries are then returned to the DSE for detailed evaluation and preparation of the final report.

C. GENERAL EVALUATOR GUIDELINES.

This section contains general information required by the individual evaluator for completion of the documentation and source listing questionnaires. Prior to the calibration test, the necessary questionnaires and answer sheets will be provided to each evaluator. In addition, each will get a package of system dependent data which includes:

(1) Software maintainability evaluation schedule.
(2) Software documentation evaluation list.
(3) Software program/module name/number list.
(4) Evaluator name/number list.

The data contained in this package will be sufficient to enable each evaluator to properly fill out the identification block of each answer sheet. The evaluators should use the Software Documentation Questionnaire and the Module Source Listing Questionnaire (or this Handbook) to obtain the specific questions.

The answers to the questions should be entered on a General Purpose (NCS) Answer Sheet. One NCS Answer Sheet should be completed for each use of a Questionnaire. If specific written comments are appropriate for a given question, they should be supplied separately to the DSE. General evaluator guidelines are summarized in Figure 3. More specific details of using the Questionnaires and Forms are explained in the following paragraphs of this section. Examples are contained in section D.

1. Response Form.

The form on which an evaluator records his responses to questions is the General Purpose (NCS) Answer Sheet. This form is processed through an optical scanner, so it is important that the appropriate circles be darkly marked and that no extraneous marks appear. Errors should be completely erased. Note that the NCS Answer Sheet contains little explanatory information since it was designed to be a general form for use by any group using questionnaires. There are three blocks on the Answer Sheet: Evaluator Name Block, Numerical Identification Block, and Evaluator Response Block.

The Evaluator Name Block contains as a minimum the last name of the evaluator; it can also contain the module name (alpha only) at the choice of the software test manager. The accuracy of this block is not as critical as the following two blocks. The suggested format with an example is given in section D.

The Numerical Identification Block contains numeric codes for the particular questionnaire type, the system, subsystem, program, module, and evaluator. The numeric codes are entered in the appropriate column fields (A through L) and the associated numbered circles are darkened. Extreme care should be taken to enter all data in this block correctly since this block is optically scanned and is effectively the only output information

General

1. Work independently.
2. Complete questionnaires in specified sequence.
3. Answer all questions--a response of A-F must be provided for each question.

Calibration Test

1. Complete questionnaires in the sequence:
   a. Software Documentation Questionnaire
   b. Module Source Listing Questionnaire
2. The specific module to be evaluated is noted in the test plan along with the list of all modules selected for evaluation. All necessary identification information will be provided in the handout.

Assessment

1. Rework responses as necessary on questionnaires completed as part of the Calibration Test.
2. Complete questionnaires on the remaining modules in the sequence specified in the evaluation handout.
3. Take care to correctly complete all information on the response forms.
4. Carefully observe the specific response guidelines contained in Part II of this handbook.

Figure 3. General Evaluator Guidelines

which uniquely associates the evaluator responses with the correct evaluator and software program information. The required format with an example is given in section D. At the option of the Software Test Manager, additional data can be entered in this block, such as the date of the evaluation (in the birthdate field) or the number of hours required to complete the answer sheet (fields O and P).

2. Response Scale

The following response scale should be used to answer each question:

A. COMPLETELY AGREE (absolutely no doubt)
B. STRONGLY AGREE
C. GENERALLY AGREE
D. GENERALLY DISAGREE
E. STRONGLY DISAGREE
F. COMPLETELY DISAGREE (absolutely no doubt)

One of these responses must be given for each question. In addition, one or more of the following standardized comment responses can be selected:

I. I had difficulty answering this question.
J. A written comment has been submitted.

The responses G and H do not currently have any meaning. The responses A to F indicate the extent to which the evaluator agrees/disagrees with the question statement. Depending on the application area and the type of question, these responses can be interpreted differently. In general, however, the response scale can be interpreted as follows:

A. COMPLETELY AGREE - There must be absolutely no doubt when using this response that the product being evaluated cannot be any better with respect to the characteristic addressed.

B. STRONGLY AGREE - This response indicates that the product being evaluated is very good and very helpful to the software maintainer.

C. GENERALLY AGREE - This response indicates that the product being evaluated is acceptable and helpful to the software maintainer.

D. GENERALLY DISAGREE - This response indicates that, although the product being evaluated is acceptable, some improvements are required to make it helpful to the software maintainer.

E. STRONGLY DISAGREE - This response indicates that the product being evaluated is unacceptable and major improvements are required before it would be helpful to the software maintainer.

F. COMPLETELY DISAGREE - There must be absolutely no doubt when using this response that the product being evaluated is unacceptable and must be completely redesigned or rewritten to be acceptable with respect to the characteristic addressed. It is emphasized that responses of A or F are in general not expected since these responses indicate a best possible or worst possible characteristic relative to software in general.

Occasionally the evaluator may find it difficult to answer a question because it "doesn't seem to apply" to the program being evaluated. Most of the time this situation arises when the particular software requirements do not involve the need for a certain software attribute; or, the intrinsic nature of the software (e.g., language used) eliminates the possibility that such a software attribute can exist. As an example, a particular module may not have any statement labels. A question statement such as "Statement labels have been named in a manner which facilitates

locating a label in the source listing" addresses a software attribute (descriptive labels) which, because the software has no labels, appears irrelevant. However, note that the software will be more understandable since no branching to labeled statements can occur. In this case, the absence of labels makes the software more maintainable, so the evaluator should give as high a response (A) as if the software had labels that were all very well named.

Sometimes the situation arises where the question implies the existence of a particular item (chart, matrix, etc.) and the question asks about the content within the item. If the item does not exist, then this indicates a serious defect (relative to maintainability) of the software. In this case, the evaluator response should be F. Questions where this situation might arise will have special response instructions included in Part II.

3. Software Documentation Questionnaire.

This questionnaire is used to evaluate the overall format and content of the documentation (not including source listings) for the computer program being evaluated. Although the information required to answer the Software Documentation Questionnaire may be spread out among several distinct documents, the primary information sources which are always considered a part of the evaluation are the program functional/ detailed design specifications and the program maintenance/operational procedures. Contractor programming conventions should also be made available. The documentation which is to be evaluated is specified to the evaluator prior to the Calibration Test.

4. Module Source Listing Questionnaire

This questionnaire is used to evaluate the overall format and content of the source listing for the program module being evaluated, and to evaluate the consistency between the module's

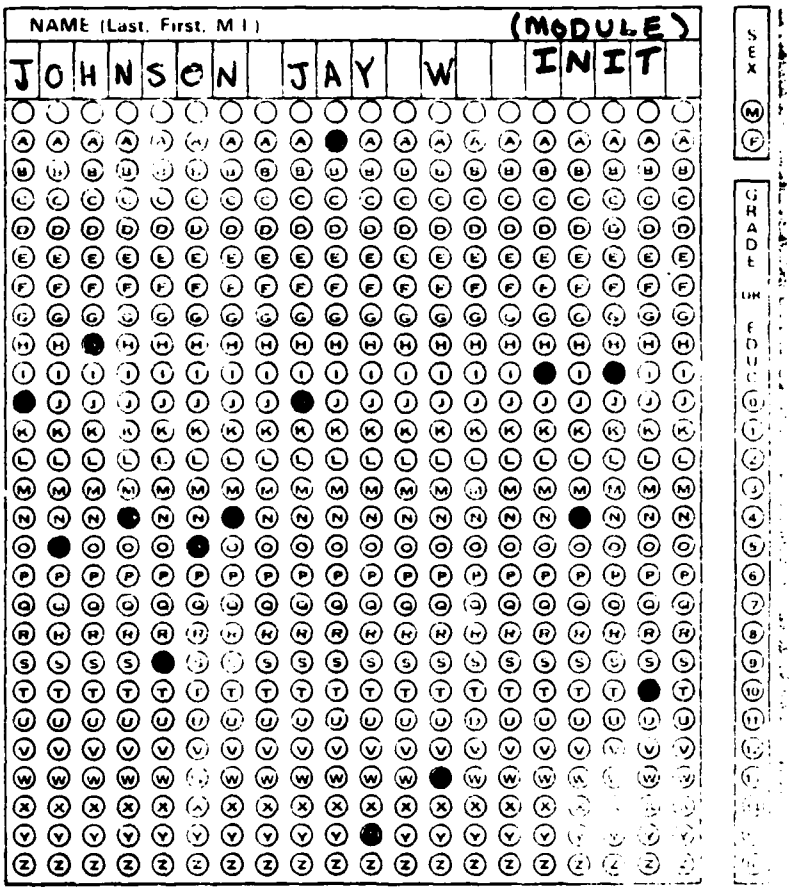
documentation and source listing. The program modules which are to be evaluated are specified to the evaluator prior to the Calibration Test.

D. EXAMPLE RESPONSE FORMS

The figures on the following pages contain response form instructions and examples as summarized below:

(1) Figure 4. Evaluator Name Block Example
(2) Figure 5. Numerical Identification Block Format
(3) Figure 6. Numerical Identification Block Example
(4) Figure 7. General Purpose (NCS) Answer Sheet

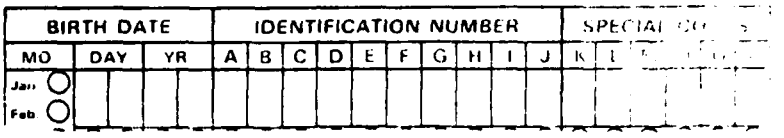
The following example is provided for completing the Name Identificiation Block on the General Purpose - NCS - Answer Sheet used with the AFTEC questionnaires.



NOTE: The SEX block and the GRADE or EDUC block need not be filled in.

Figure 4. Evaluator Name Block Example

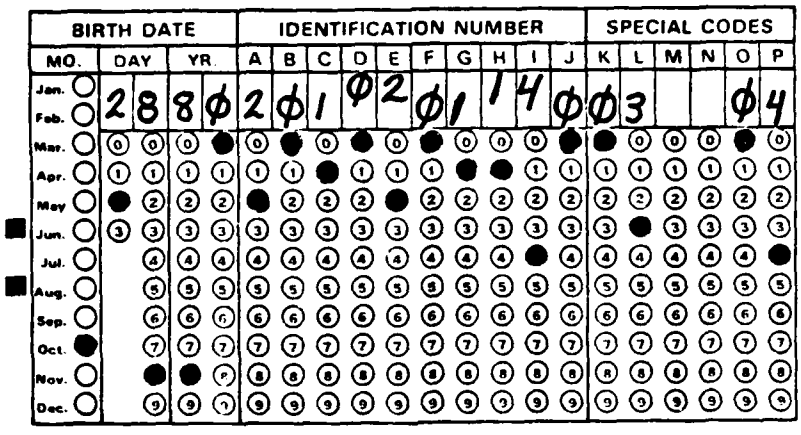
The following format is required for completing the Date and Numerical Identification Block on the General Purpose -NCS - Answer Sheet used with the AFTEC questionnaires. The correct numerical integers must be written in the appropriate fields. Extreme care should be taken in entering this data and in completely covering the associated numbered circle in each column. This numerical ID is processed by an optical scanner and is effectively the only way that the questionnaire responses can be correlated with the system/subsystem/program.../evaluator/...



Columns	Data Description	Range
Birth Date	Date Evaluation Started	
Mo.		Jan - Dec
Day		01-31
Yr.		80-99
A	Type of Questionnaire	1 (Documentation)
		2 (Source Listing)
B,C	System Code	01-99
D,E	Subsystem Code	01-99
F,G	Program Code	01-99
H,I	Module Code	01-99
J,K,L	Evaluator Code	001-999
M,N	(not used)	(Blank)
O,P	Time (hrs) to complete questions	00-99

Figure 5. Numerical Identification Block Format

The following example is provided for completing the Date and Numerical Identification Block on the General Purpose NCS Answer Sheet used with the AFTEC questionnaires.



Columns	Value	Meaning
Birth Date		
Mo. Day Yr.	Oct. 28 80	Evaluation conducted on 28 Oct 1980
A	2	Source Listing Evaluation
B,C	01	System #01
D,E	02	Subsystem #02
F,G	01	Program #01
H,I	14	Module #14
J,K,L	003	Evaluator #003 (Johnson)
O,P	04	Johnson took 4 hours to complete this questionnaire on Module #14.

Figure 6. Numerical Identification Block Example

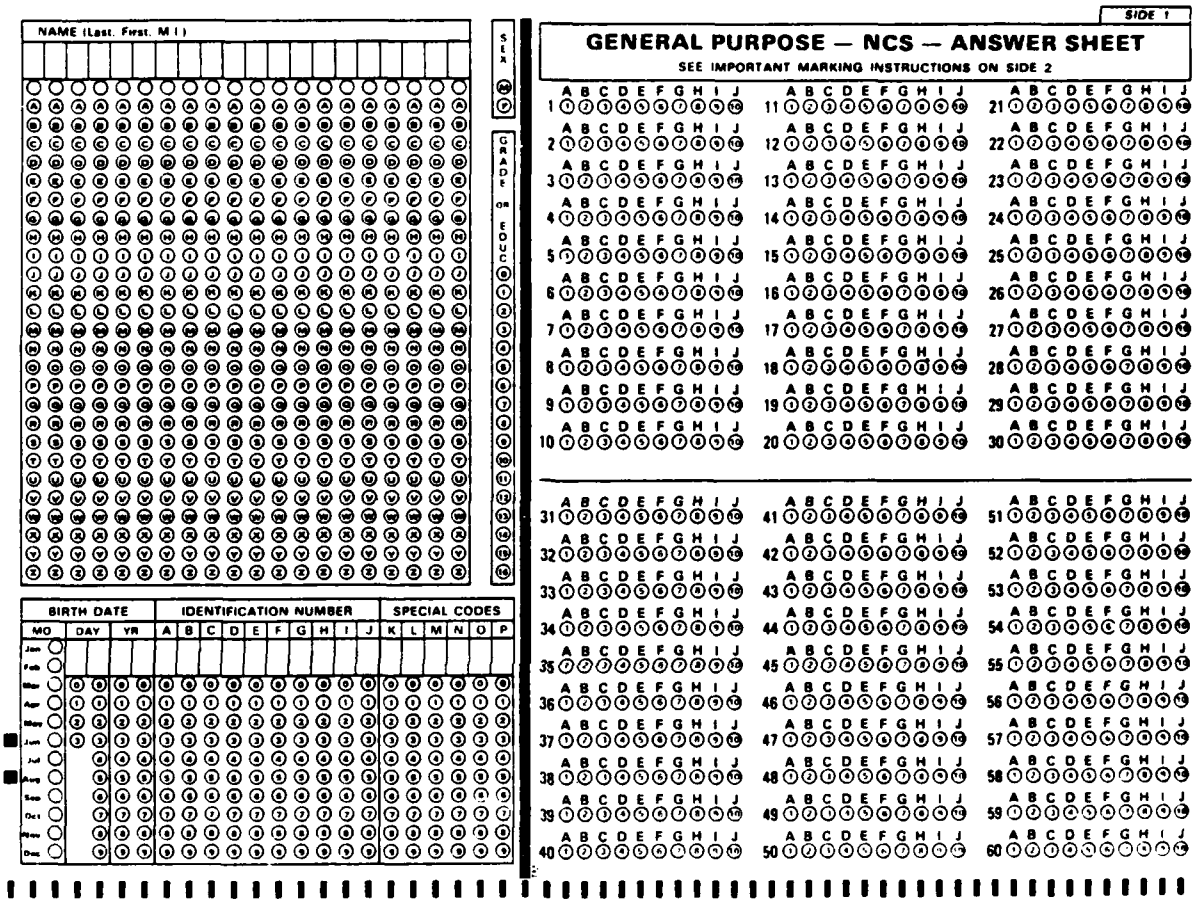


Figure 7. General Purpose (NCS) Answer Sheet

(A copyrighted form. Reprinted by Permission of National Computer Systems, Inc., Minneapolis, Minnesota.)

Note carefully the format of the question response section. The first 30 answers go in the top half of the three columns; whereas, the next 30 answers go in the bottom half of the column.

E. SUMMARY OF EVALUATION PHILOSOPHY.

This section contains a summary of the general philosophy which should guide each evaluator in answering the questionnaire.

To begin with, the evaluator will notice that the "questions" are not questions at all. They are statements describing a particularly desirable attribute of computer software documentation or coding practice. In "answering the question," the evaluator quantifies his subjective viewpoint as to what degree that desirable attribute is reflected in the system under evaluation. The average of all evaluators answers on each question then provides the basis for a statistical evaluation of the maintainability of the system.

The primary consideration for the evaluator is "How maintainable would this system be for me?" Keeping this thought in mind can simplify responses to what on the surface are non-relevant questions. There are no non-relevant questions; there are situations/systems that are more/less maintainable because of their lack of some attributes described in the questions.

It is recognized that some documentation (or source listings) may not be available to the evaluator at the time of evaluation. If this occurs, the evaluation should proceed using whatever is available; but the DSE should be prepared to brief the results tempered with the limitations under which the evaluators operated.

The definition of module for the evaluation should be well understood by all evaluators prior to the start of the evaluation. This definition should be agreed to by both the Software Test Manager and the DSE.

Unless otherwise specified, the questions should be considered as investigating both existence and quality of a characteristic; that is, the evaluator is first to determine whether or not the test factor is present in the documentation or source listing under evaluation. If he concludes that it is not present, he must

decide if that absence enhances or detracts from software maintenance on the system. If it is determined that the factor is present, the evaluator then tempers his answer to reflect the "goodness" in his judgment of that factor in the system.

Finally, bear in mind that you have been chosen for this evaluation because of your demonstrated expertise in software engineering and software support. That expertise and the professionalism you demonstrate in completing this evaluation will go far in providing the Air Force with a quality software product.

PART II

SOFTWARE MAINTAINABILITY EVALUATOR'S HANDBOOK QUESTION RESPONSE GUIDELINES

A. GENERAL.

The following sections contain information which should help clarify the intent of each question to the evaluator. Part II.B contains information on each Software Documentation Questionnaire question. Part II.C contains information on each Module Source Listing Questionnaire question.

Each question contains question terminology clarification and special response instructions as necessary.

B. SOFTWARE DOCUMENTATION QUESTIONS.

Each page within this section corresponds to a question from the Software Documentation Questionnaire. Many questions have special response instructions which should be reviewed.

QUESTION DATA SHEET

Question Number D-1

QUESTION: The documentation includes a separate part for the description of external interfaces.

CHARACTERISTIC: Modularity (format modularity).

EXPLANATIONS: Personnel working in functional areas need to have information available in one place.

EXAMPLES: An Interface Control Document (ICD). An Operator's Manual.

GLOSSARY:

Part: Section, volume, document, subsection, etc. as appropriate.

External interfaces: Program input and output data, interrupts.

SPECIAL RESPONSE INSTRUCTIONS:

Answer A if one separate part exists.

Answer B - E if the external interfaces are described in several separate parts depending upon the effectiveness of that distribution.

Answer F if no description of external interfaces is available.

QUESTION DATA SHEET

Question Number D-2

QUESTION: The documentation includes a separate part for the description of each major function.

CHARACTERISTIC: Modularity (format modularity)

EXPLANATIONS: Personnel working on a specific function should have all relevant information available in one place.

EXAMPLES: Personnel working only on the navigation function of an aircraft operational flight program should have navigation functional descriptions in one place.

GLOSSARY:

Part: Section, volume, document, subsection, etc. as appropriate.

Major function: As defined by the overview or other equivalent information: may be a component, module, etc.

SPECIAL RESPONSE INSTRUCTIONS:

Answer A if a separate part exists for each major function.

Answer B - E if each major function is described in several separate parts depending upon the effectiveness of that distribution.

Answer F if there is no description of each major function.

QUESTION DATA SHEET

Question Number D-3

QUESTION: The documentation includes a separate part for the description of the program global data base.

CHARACTERISTIC: Modularity (format modularity).

EXPLANATIONS: Personnel working with the data base should have a description of all global data items in one place.

EXAMPLES: There should be a separate part of documentation containing descriptions, types, ranges, sizes, formats, etc. of the global data items. Where lists are not complete, plans for completion should be evident.

GLOSSARY:

Part: Section, volume, document, subsection, etc. as appropriate.

Global data base: Set of all variables, constants, etc. which can be accessed by more than one program module: e.g., FORTRAN's COMMON, JOVIAL's COMPOOL, assembly's DATA MODULE, etc.

SPECIAL RESPONSE INSTRUCTIONS:

Answer A if a separate part exists or there is clearly no global data.

Answer B - E if the global data base is described in several separate parts depending upon the effectiveness of that distribution.

Answer F if no description of the global data base exists.

QUESTION DATA SHEET

Question Number D-4

QUESTION: Major parts of the documentation are essentially self-contained.

CHARACTERISTIC: Modularity (format modularity).

EXPLANATIONS: Sampling major parts of the documentation for the amount of cross referencing and the essential nature of the cross referencing should give the evaluator a general impression as to level of agreement/disagreement with the question statement. However, cross referencing for the purpose of eliminating bulky redundancies is acceptable.

EXAMPLES:

GLOSSARY:

Major parts: As essentially defined by documentation table of contents and physical structure (volumes, sections, units, etc.): might include major functions, data base description, external interfaces, test plan, conventions and standards, etc.

Self-contained: Independent, stand-alone document. Makes no cross references to other major parts of the documentation.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-5

QUESTION: The documentation has been physically separated into (sets of) volumes each with a distinct purpose.

CHARACTERISTIC: Modularity (format modularity).

EXPLANATIONS: Each (set of) volume's introduction should include an indication of what the (set of) volume's purpose is. A brief scan of the documents should give the evaluator a general impression of whether that purpose is relatively distinct, mixed, matches the stated purpose, etc. Each physically separate volume should be checked and an accumulative impression formed of the level of agreement/disagreement with the question statement.

EXAMPLES: Maintenance information should not be physically included in an operator's handbook.

GLOSSARY: Distinct purpose: These might include functional specification, detailed specification, maintenance manual, user's guide, data base description, problem reports, installation instructions, documentation plan, test plan, etc.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-6

QUESTION: The documentation indicates that each global data structure is partitioned into functionally related sets of variables.

CHARACTERISTIC: Modularity (data modularity).

EXPLANATIONS: Documentation describing the program global data base should include the set of all global data and how it has been partitioned into global data structures.

EXAMPLES: Geodetic site data should be grouped in one global data structure.

GLOSSARY:

Global data: Any variable or constant which can be accessed by more than one module of a program.

Global data structure: A particular grouping of global data variables and/or constants; e.g., FORTRAN's COMMON, JOVIAL's COMPOOL.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if there is no global data (hence no global data structures); the implication is that data coupling is decreased if there is no global data.

QUESTION DATA SHEET

Question Number D-7

QUESTION: The documentation indicates that data storage locations are not used for more than one type of data structure.

CHARACTERISTIC: Modularity (data modularity).

EXPLANATIONS: Typical multiple use of data storage locations would be dynamic memory management schemes, equivalence and overlays. Program documentation (perhaps at the module level) should describe any use of storage locations for these types of uses.

EXAMPLES: Any use of the EQUIVALENCE statement in FORTRAN, or SAME SORT, SAME AREA, or REDEFINE in COBOL should be documented.

GLOSSARY: Type: Examples of types of data structures would be integer, real, character, array of integer, array of real, array of characters, records, files, etc.

SPECIAL RESPONSE INSTRUCTIONS: If not indicated in the documentation either way, answer D.

QUESTION DATA SHEET

Question Number D-8

QUESTION: The program control flow is organized in a top down hierarchical tree pattern.

CHARACTERISTIC: Modularity (processing modularity).

EXPLANATIONS:

The documentation should include a program overview section which will likely describe the overall program control flow among modules in narrative or chart form.

Control paths which are not strictly down or up in the sense of level tend to detract from modularity because of the associated lack of independence which is introduced.

EXAMPLES:

GLOSSARY: Top down hierarchical tree pattern: One imagines a root system of a tree with each junction (node) representing a major program function or module and each branch a control path between the nodes.

SPECIAL RESPONSE INSTRUCTIONS: Answer F if there is no description (narrative or chart) of overall program control flows.

QUESTION DATA SHEET

Question Number D-9

QUESTION: The documentation indicates that program initialization processing is done by one (set of) modules(s) designed exclusively for that purpose.

CHARACTERISTIC: Modularity (processing modularity).

EXPLANATIONS:

The documentation describing the program functions and control flow should also describe how the initial program state is determined (e.g., via execution of one initialization module or not). Checks of module processing may indicate whether any initialization processing is mixed with other application functions.

It is usually better if each function, module, submodule, etc. does not handle its own global initialization. There should be one part (e.g., a few modules) of the program which is for initialization.

EXAMPLES:

GLOSSARY: Initialization: The preparatory steps required to set the initial program data and control states.

SPECIAL RESPONSE INSTRUCTIONS: If the partitioning is such that each function is performed by a set of modules and there is one module in each set expressly for initialization, then strong agreement with the question should be so indicated. If in addition, these initialization modules are all executed in preparation to any other functional activity as the first program action, then there should be essentially complete agreement with this question's statement.

QUESTION DATA SHEET

Question Number D-10

QUESTION: The documentation indicates that program termination processing is done by one (set of) module(s) designed exclusively for that purpose.

CHARACTERISTIC: Modularity (processing modularity).

EXPLANATIONS:

The documentation describing the program functions and control flow should also describe how the final program state is determined (e.g., via execution of one termination module or not). Checks of module processing may indicate whether any termination processing is mixed with other application functions.

It is best if each function, module, submodule, etc. does not handle its own termination. There should be one part (e.g., a few modules) of the program which is for termination processing.

EXAMPLES: FORTRAN's STOP statement and COBOL's STOP RUN statement could be within the processing area.

GLOSSARY: Termination: The terminal steps required to set the final program data and control states (might be due to normal/abnormal termination).

SPECIAL RESPONSE INSTRUCTIONS: If the partitioning is such that each function is performed by a set of modules and there is one module in each set expressly for termination processing, then strong agreement with the question should be so indicated. If in addition, these termination modules are executed only as a systematic program termination procedure, then there should be essentially complete agreement with this question's statement. Variations on the program termination processing should result in appropriate variations in the evaluator response depending on how much termination processing is mixed with other application functions.

QUESTION DATA SHEET

Question Number D-11

QUESTION: The documentation indicates that program I/O is done by one (set of) module(s) designed exclusively for that purpose.

CHARACTERISTIC: Modularity (processing modularity).

EXPLANATIONS:

The documentation describing the program functions and control flow should also describe how the program I/O is done (e.g., via execution of one module or more). Checks of module processing may indicate whether any I/O functions are mixed with other application functions.

It is best if each function, module, submodule, etc. does not handle its own I/O. There should be one part (e.g., a few modules) of the program which is for I/O processing.

EXAMPLES:

GLOSSARY: I/O: Input or output of program data.

SPECIAL RESPONSE INSTRUCTIONS: If the partitioning is such that each function is performed by a set of modules and there is one module in each set expressly for I/O, then appropriate agreement with the question should be so indicated. Variations on the program I/O processing should result in appropriate variations in the evaluator response depending on how much I/O is mixed with other application functions.

QUESTION DATA SHEET

Question Number D-12

QUESTION: The documentation indicates that program error processing is done by one set of modules designed exclusively for that purpose.

CHARACTERISTIC: Modularity (processing modularity).

EXPLANATIONS:

The documentation describing the program functions and control flow should also describe how the program processes any error condition (e.g., via execution of one error processing module or not). Checks of module processing may indicate whether any error processing functions are mixed with other application functions.

It is best if each function, module, submodule, etc. does not handle its own error processing unless adequate corrective measures are appropriate. There should be one part (e.g., a few modules) of the program which is for error processing.

EXAMPLES: Editing of input data should be documented.

GLOSSARY: Error processing: The steps required to set program data and control states following the detection of an error condition.

SPECIAL RESPONSE INSTRUCTIONS: If the partitioning is such that each function is performed by a set of modules and there is one module in each set expressly for error processing, then appropriate agreement with the question should be so indicated. If in addition, these error processing modules are systematically organized as an error processing function, then there should be essentially complete agreement with this question statement.

QUESTION DATA SHEET

Question Number D-13

QUESTION: Each physically separate part of the documentation includes a useful table of contents.

CHARACTERISTIC: Descriptiveness (format descriptiveness).

EXPLANATIONS: Each separately bound part of the set of documentation for this program has its own table of contents to assist in locating program information.

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Answer A to F depending upon the percentage of documents which have a useful table of contents: e.g., 100% - answer A, 0% - answer F.

QUESTION DATA SHEET

Question Number D-14

QUESTION: Each physically separate part of the documentation includes a useful glossary of major terms and acronyms unique to that document.

CHARACTERISTIC: Descriptiveness (format descriptiveness).

EXPLANATIONS: Each separately bound part of the set of documentation has its own glossary of major terms and acronyms to assist in clarifying other documentation.

EXAMPLES:

GLOSSARY:
Acronym: A term formed by the initial letter(s) of a series of one or more words.
Example: FORTRAN = FORmula TRANslation.

SPECIAL RESPONSE INSTRUCTIONS: Answer A to F depending upon the percentage of documents which have a useful glossary: e.g., 100% - answer A, 0% answer F.

QUESTION DATA SHEET

Question Number D-15

QUESTION: Each physically separate part of the documentation includes a useful index.

CHARACTERISTIC: Descriptiveness (format descriptiveness).

EXPLANATIONS: Each separately bound part of the set of documentation has its own index to assist in locating information.

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Answer A to F depending upon the percentage of documents which have a useful index: e.g., 100% - answer A, 0% - answer F.

QUESTION DATA SHEET

Question Number D-16

QUESTION: It is easy to locate specific information within the documentation.

CHARACTERISTIC: Descriptiveness (format descriptiveness).

EXPLANATIONS: The evaluator should repeatedly conceptualize the need for locating a specific piece of information that might be needed for maintenance, and then check the documentation for the effort required to locate the information.

EXAMPLES: One piece of frequently needed information might be the contents of the parameter list. Another might be a list of what modules call another module. The evaluator should consider some specific piece of information and assess the ease of locating that information.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-17

QUESTION: The documentation includes a useful version description document.

CHARACTERISTIC: Descriptiveness (format descriptiveness).

EXPLANATIONS: Some document should be readily available which describes the current operational version of each configuration controlled program. In hierarchical library systems, documents should be available describing each level of the library.

EXAMPLES:

GLOSSARY: Version description document: A document which describes the version of each computer program.

SPECIAL RESPONSE INSTRUCTIONS: If the documentation is a baseline (original version or an all-encompassing rewrite), the evaluator should answer A.

QUESTION DATA SHEET

Question Number D-18

QUESTION: A useful master list is available which identifies all software documentation.

CHARACTERISTIC: Descriptiveness (format descriptiveness).

EXPLANATIONS: A reference list should be available in one overview document or in a separate document which lists at least all delivered program-related documentation by name and description; if several programs are part of the software developement effort, then the list should contain information on all programs.

EXAMPLES:

GLOSSARY: Master list: This may be a reference list in one overview document, or a separate document itself.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-19

QUESTION: Any dynamic allocation of resources (storage, timing, priority, hardware services, etc.) is explained in the documentation.

CHARACTERISTIC: Descriptiveness (constraints descriptiveness).

EXPLANATIONS: The documentation describing the functional/detailed program specifications should include a section which explains what dynamic allocation features are used by the program. These features may be considered necessary depending upon the application, but all are considered to increase the effort required to maintain a program.

EXAMPLES: The most common dynamic allocation feature is probably storage allocation. There may be allocation routines which must be called to get or release memory. Also, the priority scheme or timing allocation for particular "rate" groups may depend upon certain phases of a mission and dynamically change on that basis. Likewise assignment of control of tape drives, discs, communication lines or other hardware may be done in some dynamic manner.

GLOSSARY: Dynamic allocation: Any assignment of a resource which is (or can be) done during the execution of a program; contrasts with "static allocation" which implies the resource assignment remains fixed throughout program execution.

SPECIAL RESPONSE INSTRUCTIONS:

Answer A if it is clear there is no dynamic allocation of resources for this program.

Answer B - F otherwise.

QUESTION DATA SHEET

Question Number D-20

QUESTION: Timing requirements for each major function of the program are explained in the documentation.

CHARACTERISTIC: Descriptiveness (constraints descriptiveness).

EXPLANATIONS: The allocated time for each major function operating in a real-time environment should be described in the documentation. In addition, the timing relationships among major functions, or the framing scheme, should also be described and readily available.

EXAMPLES:

GLOSSARY: Major function: The program overview, hierarchical chart, etc. will ordinarily define what major function (and its components) means; it usually will correspond to a module or group of modules as defined for a given program evaluation.

SPECIAL RESPONSE INSTRUCTIONS:

Answer A if it is clear that this program has no timing requirements (e.g., is non-real time).

Answer B - F otherwise.

QUESTION DATA SHEET

Question Number D-21

QUESTION: Storage requirements for each major function of the program are explained in the documentation.

CHARACTERISTIC: Descriptiveness (constraints descriptiveness).

EXPLANATIONS: Allocated storage requirements for each major function should be described in the documentation. Even if a program does not have any "critical" storage requirements, there should be an explanation in the documentation covering the program's environment.

EXAMPLES:

In a program operating in a paged storage environment, the page limitations (number of pages, boundary requirements, etc,) should be described.

For programs operating in a resident/non-resident environment, relationships to the roll-in area requirements should be described.

GLOSSARY: Major function: The program overview, hierarchical chart, etc. will ordinarily define what major function (and its components) means; it usually will correspond to a module or group of modules as defined for a given program evaluation.

SPECIAL RESPONSE INSTRUCTIONS: Answer F if there is no explanation of the storage requirement(s).

QUESTION DATA SHEET

Question Number D-22

QUESTION: The inputs to each module are explained in the documentation.

CHARACTERISTIC: Descriptiveness (module descriptiveness).

EXPLANATIONS: Input parameters passed via parameter packages or argument lists and global data used by each module as input should be described.

EXAMPLES: The documentation for a trigonometric subroutine describes what data is input (an angle), the form (in radians), limitations ($0 \leq$ angle $\leq \pi/2$), and how it is input (passed as a single precision real number in the first parameter).

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-23

QUESTION: The processing done by each module is explained in the documentation.

CHARACTERISTIC: Descriptiveness (module descriptiveness).

EXPLANATIONS: The algorithm(s) which generate the outputs from the inputs should be described in the documentation.

EXAMPLES: A trigonometric function should have a description of the function itself, the algorithm used, and any limitations.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-24

QUESTION: The outputs from each module are explained in the documentation.

CHARACTERISTIC: Descriptiveness (module descriptiveness).

EXPLANATIONS: Output parameters passed via parameter or argument lists and global data altered by each module should be described.

EXAMPLES: The documentation for an inverse trigonometric subroutine describes what data is output (an angle), the form (in radians), and how it is output (passed as a double precision real number in the second parameter).

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-25

QUESTION: Special processing considerations (error, interrupt, etc.) of each module are explained in the documentation.

CHARACTERISTIC: Descriptiveness (module descriptiveness).

EXPLANATIONS: Any special considerations, such as the different types of errors possible and their effects, the effects of interrupts and the effects of other asynchronous events should be described.

EXAMPLES: In a message processing program, processing limitations may cause loss of an incoming character. The documentation for the input handler should describe this condition and its response to it.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-26

QUESTION: There is a flow chart (or equivalent) for each module which adequately illustrates the inputs, general processing, and outputs for the module.

CHARACTERISTIC: Descriptiveness (module descriptiveness).

EXPLANATIONS: Some form of functionally-oriented presentation of each of the program components should be available in the documentation. This could take the form of flowcharts, Process Design Language (PDL) with functional commentary, Hierarchical Input-Processing-Output (HIPO) charts, etc.

EXAMPLES:

GLOSSARY: Flowchart (or equivalent): A logic flow diagram in which symbols are used to represent operations, data, flow, equipment, etc. Examples are: FORTRAN flowchart, Process Design Language (PDL), Hierarchical Input-Processing-Output (HIPO) chart, etc.

SPECIAL RESPONSE INSTRUCTIONS: Answer F if module flowcharts (or equivalent) do not exist.

QUESTION DATA SHEET

Question Number D-27

QUESTION: Program initialization and termination processing is explained.

CHARACTERISTIC: Descriptiveness (external interface descriptiveness).

EXPLANATIONS: The documentation should cover both the data and the steps required to initialize the operation of this program within the system and to effect both normal and abnormal termination of the program.

EXAMPLES: An Operational Flight Program may have no termination procedures short of power off; however, most such programs determine the source of the power outage, and whether any memory locations need be protected, etc. Such considerations should be documented.

GLOSSARY:

Initialization: The preparation steps required to set the initial program data and control states.

Termination: The terminal steps required to set the final program data and control states (might be due to normal/abnormal termination).

SPECIAL RESPONSE INSTRUCTIONS: The evaluator should study both initialization and termination processing explanations. The response A-F should reflect overall how well both have been explained.

QUESTION DATA SHEET

Question Number D-28

QUESTION: Recovery from externally generated error conditions which could affect the program is explained.

CHARACTERISTIC: Descriptiveness (external interface descriptiveness).

EXPLANATIONS: The documentation should include an explanation of overall error processing. This description should include a description of the recovery of the program from error conditions generated external to the program, but affecting its capability to function. In most cases, this explanation will concern the recovery from lack of or bad input data or parameters to the program.

EXAMPLES:

GLOSSARY: Recovery: The procedures taken to report/correct some program failure (resulting from an external error condition in this case and probably recognized as bad input data).

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-29

QUESTION: The process of recovering from internally generated error conditions is explained.

CHARACTERISTIC: Descriptiveness (external interface descriptiveness).

EXPLANATIONS: The documentation should include an explanation of overall error processing. This description should include a description of the recovery of the program from error conditions encountered within the program and not directly caused by the environment external to the program.

EXAMPLES: The documentation explains that, in cases where a divide by zero is possible, a check is made of the divisor and alternate processing is instituted to recover from the error.

GLOSSARY: Internal error condition: Any algorithm failure due to processing of internally defined data.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-30

QUESTION: Input of program data is explained.

CHARACTERISTIC: Descriptiveness (external interface descriptiveness).

EXPLANATIONS: The documentation should describe what data is input, the form of the data, any limitations on the data, and how it is input.

EXAMPLES:

1. Card deck or card deck image input: Line-by-line description of input, giving format, range or limitations of each data field, type (numeric or alphanumeric, integer or floating point), etc.
2. Multiplex Bus: Description of all data structures to be received from the bus, giving source and timing of data blocks (such as a block received from the inertial navigation system once per second), the sequence, definition, and scale factors of the parameters in a block, etc.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-31

QUESTION: Output of program data is explained.

CHARACTERISTIC: Descriptiveness (external interface descriptiveness).

EXPLANATIONS: The documentation should describe what data is output, the form of that data, and how it is output.

EXAMPLES: Complete description of the program output, be it:
1. Listing (printout),
2. CRT display (data displayed on a Heads Up Display [HUD]), or
3. Mux Bus, etc.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-32

QUESTION: There is a useful set of charts which show the general program control and data flow hierarchy among all modules.

CHARACTERISTIC: Descriptiveness (internal interface descriptiveness).

EXPLANATIONS: Whatever method is used to present the flow, the presentation should be understandable and complete.

EXAMPLES: The documentation should include a set of system flowcharts, Process Design Language (PDL), Hierarchical Input-Processing-Output (HIPO), etc. which show the program control and data flow either together or separately.

GLOSSARY: Chart: Flowchart, Process Design Language (PDL), Hierarchical Input-Processing-Output (HIPO) chart, etc.

SPECIAL RESPONSE INSTRUCTIONS: Answer F if no set of charts exists.

QUESTION DATA SHEET

Question Number D-33

QUESTION: There is a master list (chart, table, section, etc.) identifying where each global variable is used.

CHARACTERISTIC: Descriptiveness (internal interface descriptiveness).

EXPLANATIONS: A part of the documentation should be a master list identifying where each global variable is used. This list contains information used by maintainers of all modules and it is important they use the same list.

EXAMPLES: In many programming systems, an automated global data cross-reference report may be generated.

GLOSSARY: Global variable: Any variable which can be accessed by more than one module of a program; global constants should also be identified.

SPECIAL RESPONSE INSTRUCTIONS: Answer F if no master list or its equivalent exists. Answer A if it is clear that no global variables exist in the progra...

QUESTION DATA SHEET

Question Number D-34

QUESTION: The global variable master list includes information about each global variable such as type, range, scaling, units, etc.

CHARACTERISTIC: Descriptiveness (internal interface descriptiveness).

EXPLANATIONS: The documentation should contain a separate data base description in which all global data is described to include information on type, range, etc. This list is important in that it contains information used by maintainers of all modules.

EXAMPLES:

GLOSSARY: Global variable: Any variable which can be accessed by more than one module of a program; global constants should also be identified.

SPECIAL RESPONSE INSTRUCTIONS: Answer F if no master list or its equivalent exists. Answer A if it is clear that no global variables exist in the program.

QUESTION DATA SHEET

Question Number D-35

QUESTION: The use of any complex mathematical model (technique, algorithm) is explained in the documentation.

CHARACTERISTIC: Descriptiveness (math model descriptiveness).

EXPLANATIONS: The documentation should contain details on the use of any complex algorithm to include input requirements and limitations.

EXAMPLES: The documentation for a numerical integration algorithm might specify that a minimum number of intervals be selected for a specified result accuracy.

GLOSSARY: Complex mathematical model: e.g., Fourier transform, Laplace transform, numerical integration/differentiation scheme, control theory algorithm, statistical technique, etc.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if it is clear that there are no complex mathematical models (techniques, algorithms) used in the program.

QUESTION DATA SHEET

Question Number D-36

QUESTION: The documentation on each complex mathematical model includes information such as a derivation, accuracy requirements, stability considerations and references.

CHARACTERISTIC: Descriptiveness (math model descriptiveness).

EXPLANATIONS: The documentation should contain enough detailed explanation or cross-references to allow the maintainer to modify the algorithm or its implementation and be aware of the implications or be able to locate references which make the implications clear.

EXAMPLES: A numerical algorithm that depends on double precision processing should have a description of the implications to accuracy if single precision were to be substituted.

GLOSSARY: Complex mathematical model: e.g., Fourier transform, Laplace transform, numerical integration/differentiation scheme, control theory algorithm, statistical technique, etc.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if it is clear that there are no complex mathematical models (techniques, algorithms) used in the program.

QUESTION DATA SHEET

Question Number D-37

QUESTION: It appears that a useful set of standards has been followed for the development of the documentation.

CHARACTERISTIC: Consistency (format consistency).

EXPLANATIONS: Consistent documentation means that the maintainer can spend less time learning the organization of the documentation and more time learning the content. The documentation should be scanned for adherence to standards.

The evaluator may know in advance that documentation standards were generated and he can see that they were followed.

The evaluator may not know in advance but may be able to tell from the organization of diverse parts of the documentation that standards were available and followed.

Confusing documentation organization indicates misuse or no use of documentation standards.

EXAMPLES: Following either contractor standards developed locally or universal standards (e.g., ANSI FORTRAN) which help understandability. Usually the format consistency of the documentation indicates how much a standard/convention, etc. has been followed.

GLOSSARY: Standards: Procedures, rules, and conventions used for prescribing disciplined program design and implementation.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-38

QUESTION: It appears that a set of standards has been followed for the construction of all (program and module) flowcharts (or equivalent).

CHARACTERISTIC: Consistency (format consistency).

EXPLANATIONS: The flowcharts of the program and its modules should be scanned for conventional use of symbols, labeling consistency, etc. There may be a stated standard (e.g., ANSI FORTRAN) against which the flowcharts may be compared.

EXAMPLES: The documentation contains a section describing the flowcharting methodology and it is clear from the flowcharts that the methodology has been followed.

GLOSSARY:

Standards: Procedures, rules and conventions used for prescribing program design and implementation.

Flowchart (or equivalent): A logic flow diagram in which symbols are used to represent operations, data, flow, equipment, etc. In the broad sense, would include FORTRAN flowchart, Process Design Language (PDL), Hierarchical Input-Processing-Output (HIPO), etc.

SPECIAL RESPONSE INSTRUCTIONS: Answer F if there are no flowcharts (or equivalents).

QUESTION DATA SHEET

Question Number D-39

QUESTION: Documentation of each major functional part of the program follows the same format.

CHARACTERISTIC: Consistency (format consistency).

EXPLANATIONS: Each major functional area of a program should have the same documentation format as far as is practicable in order to aid understandability.

EXAMPLES: An airborne computer may contain major modules dedicated, for instance, to navigation, bombing, and air-to-air. Each of these modules would need input, output, and processing sections. All input sections should be similar; all output sections should be similar, etc.

GLOSSARY: Major functional part: The program overview, hierarchical chart, etc. will ordinarily define what major function (and its components) means; it usually will correspond to a module or group of modules as defined for a given program evaluation.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-40

QUESTION: The format of the documentation reflects the organization of the program.

CHARACTERISTIC: Consistency (format consistency).

EXPLANATIONS:

Program parts are easier to maintain if the documentation has separate sections to describe each of those parts. This simplifies looking for details concerning those program parts.

There can be other considerations which may influence the evaluator in responding to this question. What is desired is basically the evaluator's general impression as to the usefulness of the documentation format in understanding the overall program organization.

EXAMPLES: Major program functions, the program data base, etc. might have separate sections. The descriptions of how the program is designed to be tested should be reflected in the format of the documentation such as providing sections for unit test procedures and sample test data if appropriate.

GLOSSARY: Organization of the program: Design of the program as components, modules, global data base, units, segments, etc.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-41

QUESTION: It appears that programming conventions have been established for the interfacing of modules.

CHARACTERISTIC: Consistency (design consistency).

EXPLANATIONS: Module interface design is extremely important; improper interfacing can lead to many hidden errors. Program design conventions should be documented. In addition, the module descriptions can be scanned to determine whether such conventions have been established and/or followed. The establishment of linkage conventions is especially important for assembly language modules.

EXAMPLES: Inputs and outputs, both argument type and global data type, require coordination between the sender(s) and the receivers(s). Such coordination requires explicit description of all attributes of each such variable and should be listed in an interface control document.

GLOSSARY:

Interfacing of modules: The passing of control, data, or services between two or more modules.

Convention: Agreed method or form of presentation to provide consistency and understanding.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if it is clear that there is no interfacing between any of the modules.

QUESTION DATA SHEET

Question Number D-42

QUESTION: It appears that programming conventions have been established for I/O processing.

CHARACTERISTIC: Consistency (design consistency).

EXPLANATIONS:

Program I/O processing is the interface of the program to the rest of its environment.

The module descriptions should be scanned to determine if any particular design consistency/conventions have been followed for program I/O processing.

EXAMPLES: One module or set of modules should be clearly identified as interfacing the computational system to the real world. All attributes of all inputs and all outputs should be clearly identified. This data is essential to all personnel interfacing with any I/O data, whether externally (to/from real world) or internally (to/from processing routines).

GLOSSARY: I/O processing: The physical input or output of program data.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-43

QUESTION: It appears that design conventions have been established for error processing.

CHARACTERISTIC: Consistency (design consistency).

EXPLANATIONS: Centralized processing of error conditions generally improves the maintainability of a program. Under such centralized error processing, any module which communicates an error condition to an error processing routine must do so properly. Therefore, error processing procedures must be documented and followed.

EXAMPLES: An error type is generated and passed to the error processing routine(s). The routine generating the error type "knows" that the error processing routine will handle it properly when both parties have followed the documented procedures.

GLOSSARY: Error processing: The procedure followed after a program failure due to some recognized error condition.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-44

QUESTION: A naming convention for modules appears to have been used.

CHARACTERISTIC: Consistency (design consistency).

EXPLANATIONS: Naming conventions help to describe processing and input/output. The maintenance programmer should be able to easily recognize calls to processes external to the module being changed. Although the listing may not be available to confirm conventions, the documentation should contain standards or conventions for naming yet-to-be-designed modules.

EXAMPLES: All routine names begin with "SUB" (for subroutine) or "XR" (for external-routine).

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-45

QUESTION: A naming convention for global variables appears to have been used.

CHARACTERISTIC: Consistency (design consistency).

EXPLANATIONS: Naming conventions help to describe processing and input/output. The maintenance programmer should be able to recognize global variables easily, since extra caution must be used when making changes which deal with data which is either generated or used outside the module being changed. Although the listings may not be available to confirm conventions, the documentation should contain standards or conventions to be followed during programming.

EXAMPLES: All variables which are global variables have names beginning with "XG" (external-global); no other type of variable name begins with that letter combination.

GLOSSARY: Global variable: Any variable or constant which can be accessed by more than one module of a program.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if there are no global variables.

QUESTION DATA SHEET

Question Number D-46

QUESTION: The terminology used in the documentation to describe the program is easily understood.

CHARACTERISTIC: Simplicity (format simplicity).

EXPLANATIONS: The general use of English and program terms should be simple, straightforward, easily understood; any terms or acronyms needing to be clarified should be defined prior to use and included in a glossary for reference.

EXAMPLES: A program that calculates MTBF should define Mean Time Between Failures - what the acronym means plus how the figure is calculated.

GLOSSARY: Terminology: Technical or special terms relevant to this particular computer system.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-47

QUESTION: The documentation is physically organized as a systematic description of the program from levels of less detail to levels of more detail.

CHARACTERISTIC: Simplicity (format simplicity).

EXPLANATIONS: Generally, the documentation produced during a software development effort should successively describe requirements, preliminary design, detailed design, operation/maintenance manual, test plan, etc. This will reflect a natural progression of program description from levels of less detail to levels of more detail.

EXAMPLES: Within any given documentation product, e.g., the detailed design specification, there should be a sequential progression from descriptions of less detail (e.g., overview) to descriptions of more detail (e.g., module design).

GLOSSARY: Physically organized: The documents, volumes, chapters, sections, etc.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-48

QUESTION: Each part (sentence, paragraph, subsection, section, chapter, volume, etc.) of the documentation tends to express one central idea.

CHARACTERISTIC: Simplicity (format simplicity).

EXPLANATIONS: All documentation should be scanned. If the documentation has been written in a simple understandable manner, then more than likely each part will address one primary topic (and subparts, one primary subtopic, etc.). The descriptions will be simple and to the point.

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-49

QUESTION: The amount of cross referencing among parts of the documentation contributes to the understandability of the program description.

CHARACTERISTIC: Simplicity (format simplicity).

EXPLANATIONS: Some parts of the documentation may use cross referencing well while other parts may not. The evaluator should study the documentation until a reasonably well-founded overall impression is formed.

EXAMPLES: A narrative description of a file layout cross referenced to a figure graphically displaying the file is good. A simple reference to the figure with no narrative is not.

GLOSSARY: Cross reference: A note, statement, section number, etc. which directs a reader from one part of the documentation to another part.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-50

QUESTION: The documentation indicates that the program source language is a high order language (HOL).

CHARACTERISTIC: Simplicity (design simplicity).

EXPLANATIONS: Even though the system design dictates a non-HOL, a HOL is desirable from a maintainablity standpoint. Less knowledge of internal machine operating characteristics is required to maintain a HOL program.

EXAMPLES: A particular communication processor program is better designed in assembly language due to the nature of bit manipulation requirements; however, assembly language programs are harder to maintain due to the machine dependency of assembly languages and the specialized knowledge required to maintain them.

GLOSSARY: High order language: A programming language that does not reflect the structure of any one given computer or that of any given class of computers: Non-assembly, non-micro code, non-machine; e.g., FORTRAN, JOVIAL, PL/I, PASCAL, etc.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if the program source language is completely HOL. Answer F if the program source language is completely non-HOL. Answer in the range B to E if the source language is a mix of HOL and non-HOL by approximate percentage:

B - $\geq$ 80%
C - $\geq$ 60%
D - $\geq$ 40%
E - $\geq$ 20%

QUESTION DATA SHEET

Question Number D-51

QUESTION: The documentation indicates that the use of recursive/reentrant programming techniques is not excessive.

CHARACTERISTIC: Simplicity (design simplicity).

EXPLANATIONS:

The documentation should identify within a general "program design considerations" section or the individual module description sections whether recursion or reentrancy is to be utilized. Many languages (or at least a particular implementation of a compiler) do not allow recursive and/or reentrant code. Some languages (e.g., stack-oriented languages like Pascal) allow recursion as a natural language capability.

The evaluator should get an overall impression of how much recursion/reentrant programming is a part of the overall program design. If done with care, some use of recursion or reentrancy can simplify the overall program design even though the particular modules which are recursive/reentrant will probably be harder to maintain because of those concerns.

EXAMPLES: Utility modules generally use these techniques.

GLOSSARY:

Recursive programming techniques: The use of operations which are defined in terms of themselves: a recursive module is one which uses a call to itself within the body of the code.

Reentrant programming technique: The technique of interrupting a program module at any point by another user and then resuming execution at the point of interruption. A reentrant module is one which can be concurrently used by more than one user.

Excessive: Detracts from simplicity.

SPECIAL RESPONSE INSTRUCTIONS:

If the documentation does not indicate, then:

Answer A if the language does not allow such techniques (example: COBOL).

Answer F if the language does allow such techniques (example: ALGOL or assembly language).

QUESTION DATA SHEET

Question Number D-52

QUESTION: The documentation indicates that each program module is designed to perform only one major function.

CHARACTERISTIC: Simplicity (design simplicity).

EXPLANATIONS: From the standpoint of simplicity, it would be easy to maintain a program in which each module performs only one function. Even if each module (or nearly each) performs only one major function and possibly one or two related functions, the program should still be simple and easy to maintain.

EXAMPLES: A print module may make a decision as to where to return in a program based upon the data printed. This may detract little from the simplicity; however, it would preclude an A answer.

GLOSSARY:
Function: A sub-division of a process.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if each module performs just one function. Answer B-F based on the proportion of modules which perform more than one function, e.g., B if 10% or less to F if 90% or more.

QUESTION DATA SHEET

Question Number D-53

QUESTION: The documentation indicates that resource (storage, timing, tape drives, disks, consoles, etc.) allocation is fixed throughout program execution.

CHARACTERISTIC: Simplicity (design simplicity).

EXPLANATIONS: Dynamic allocation tends to increase the level of complexity of a module, thereby making maintenance more difficult and time-consuming. The sharing or dynamic reassignment of resources should be a highlight of a section describing special processing (control) considerations, memory allocation, timing requirements by mission phase, etc. As another recourse, the evaluator can check the individual module descriptions for possible mention of any dynamic resource allocation.

EXAMPLES:

GLOSSARY:

Resource allocation: The assignment of a particular resource to a particular program task, function, module, etc.

Fixed: Is not reassigned from initialization to termination or reinitialization of the program.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if resource allocation is fixed throughout the entire program or is controlled by the operating system (i.e., transparent to the programmer).

QUESTION DATA SHEET

Question Number D-54

QUESTION: The documentation indicates that the control flow among modules is easy to follow.

CHARACTERISTIC: Simplicity (design simplicity).

EXPLANATIONS: The documentation should include narrative or a hierarchical flowchart which gives a clear, concise, easily understood general overview of the sequence in which modules (and perhaps submodules) are invoked and what controls that sequence.

EXAMPLES:

GLOSSARY: Control flow among modules: Which modules call and are called by other modules.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if all modules are entirely independent.

QUESTION DATA SHEET

Question Number D-55

QUESTION: The timing scheme designed for the program is easily understood from the documentation.

CHARACTERISTIC: Simplicity (design simplicity).

EXPLANATIONS: The program documentation should include a separate section which describes overall timing requirements and the timing scheme designed to satisfy those requirements. This description should be clear, concise, and easily followed.

EXAMPLES:

GLOSSARY: Timing scheme: Time slicing, time sharing, priority levels, rate groups, etc. as applied to the overall sequencing and execution of program functions.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if there are clearly no special timing considerations.

QUESTION DATA SHEET

Question Number D-56

QUESTION: The program is designed so that modules are not interrupted during execution.

CHARACTERISTIC: Simplicity (design simplicity).

EXPLANATIONS: Whenever special processing is required to handle the possibility of being interrupted, a higher level of complexity will exist in a module.

EXAMPLES:

GLOSSARY: Interrupted: Execution is suspended without the knowledge of the module being suspended.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-57

QUESTION: It is evident from the documentation that a knowledge of mathematics beyond basic algebra is not required to understand the mathematical functions performed by the program.

CHARACTERISTIC: Simplicity (design simplicity).

EXPLANATIONS: There may be a few complex functions, but on the average most of the functions require no mathematics beyond basic algebra. In this case, the evaluator might generally or strongly agree with the question statement. If there appears to be many complex functions, the evaluator may want to generally or strongly disagree with the question statement.

EXAMPLES:

GLOSSARY: Basic algebra: Functions (including trigonometric and geometric functions), equations, polynomials (including series), graphing of functions, basic manipulations, etc.; excludes calculus, differential equations, Fourier transforms, statistical algorithms, etc.

SPECIAL RESPONSE INSTRUCTIONS: The evaluator should respond on the basis of overall program considerations.

QUESTION DATA SHEET

Question Number D-58

QUESTION: A numbering scheme has been adopted which allows for easy addition or deletion of narrative parts of the documentation.

CHARACTERISTIC: Expandability (format expandability).

EXPLANATIONS: Computer program documentation can be voluminous and subject to frequent changes due to program modifications and format requirement alterations. This question seeks to determine if:

a) A numbering convention has been established for formatting the documentation; and,

b) the format enhances:

1 identifying volumes, sections, and paragraphs; and pages, and,

2 adding and deleting information without generating attendant rippling effects throughout the rest of the document. Determine if a numbering scheme has been established. Assess the ease with which a volume/section/paragraph/page can be located and the extent to which a change in document content will affect the numerical identifiers of other parts of the document.

EXAMPLES: Consecutive numbering of pages makes it difficult to add/delete pages. Use of a hierarchical numbering system to number pages by section reduces the number of succeeding pages affected by changing the contents of a section.

GLOSSARY: Number scheme: A formatting convention used to facilitate identifying some part of a document.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-59

QUESTION: Graphic materials (figures, charts, lists, etc.) are physically separate (e.g., on separate pages) from narrative description.

CHARACTERISTIC: Expandability (format expandability).

EXPLANATIONS: Graphic materials should always be on separate pages. Changes in narrative are more easily typed when narrative and graphic materials are not co-located on the same page.

EXAMPLES:

GLOSSARY: Graphic materials: Tables, figures, equations.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-60

QUESTION: A numbering scheme has been adopted which allows for easy addition or deletion of graphic materials.

CHARACTERISTIC: Expandability (format expandability).

EXPLANATIONS: Graphic materials in computer program documentation can be subject to frequent changes due to program or requirement modifications. A suitable numbering scheme should have been established such that graphic materials can easily be identified and added/removed without having a rippling effect on other numbered items in the document. It should be determined if a numbering scheme has been established. The ease with which graphic materials can be located and the extent to which adding or deleting an item affects the assigned identifiers of other items should be assessed.

EXAMPLES: Consecutive numbering of figures across major sections requires more changes when adding or deleting figures than numbering consecutively within a major section.

GLOSSARY:

Numbering scheme: A formatting convention used to facilitate identifying some part of a document.

Graphic materials: Items such as tables, figures, and equations.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-61

QUESTION: The program timing scheme appears to be flexible enough to allow for modifications (e.g., reorganization, addition, deletion of functional parts).

CHARACTERISTIC: Expandability (design expandability).

EXPLANATIONS: In many applications, specific program functions must be performed at periodic intervals, within predetermined time intervals, or at a definite point in time. This question seeks to determine the extent to which the program's timing scheme restricts desired changes to a program's design.

EXAMPLES: A function that must be performed every 10 microseconds will conflict with the design of a different function requiring 10 or more microseconds of uninterrupted processing.

GLOSSARY:

Timing scheme: A convention based on wall clock time or processor clock time that controls execution of a program's functions.

Flexible: Modifiable.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if there are no required program timing considerations.

QUESTION DATA SHEET

Question Number D-62

QUESTION: There is a reasonable time margin for each major program function (rate group, time slice, priority level, etc.).

CHARACTERISTIC: Expandability (design expandability).

EXPLANATIONS: Program functions should be designed such that required timing constraints are met with "room to spare." Too little reserves limit the ability to add processes to a function. Too much reserve, on the other hand, may indicate processing inefficiency due to resource underutilization.

EXAMPLES: A program function requiring a periodic 5 millisecond time slice is allocated a dedicated 20 millisecond time slice. The timing margin for this function is 75%.

GLOSSARY:

Timing margin: A percentage of the time allocated to a process that is still available for use; calculated by the ratio of spare time to the total time frame.

Program function: A generic term used to reference one or more program processes.

Time slice: A predetermined period of processer time.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if the program has no timing requirements because timing margins will not be of any concern (e.g., program in non-real time). Also answer A if each timing margin is at least 25%.

QUESTION DATA SHEET

Question Number D-63

QUESTION: Documentation narrative explains the procedures for altering basic data storage sizes.

CHARACTERISTIC: Expandability (design expandability).

EXPLANATIONS: How to alter the capacity of data storage is not always obvious. Very often, storage has been judiciously allocated to interface with various portions of the program. Documentation narrative should not only describe how to alter basic data storage sizes, but should also identify those interfaces which might be impacted by such changes.

EXAMPLES: Creating a new variable in the middle of a labeled common region can affect all program processes that use that storage area.

GLOSSARY: Basic data storage sizes: The size of program data structures upon which program processing depends; the structure may be an array, a table, space allocated by an assembly directive, etc.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-64

QUESTION: The program has been designed to allow for an increase in storage utilized before storage capacity is exceeded.

CHARACTERISTIC: Expandability (design expandability).

EXPLANATIONS: Over time, the amount of data storage space required for program applications almost always increases. A program should be designed so that additional storage allocations can be made without the need for program design or hardware modification.

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Answer A if at least 25% of the storage capacity is available for future use.

QUESTION DATA SHEET

Question Number D-65

QUESTION: Those modules dependent upon data structure sizes are identified.

CHARACTERISTIC: Expandability (design expandability).

EXPLANATIONS: Changing the definition of a data structure will invariably impact the modules that use it. Therefore, the documentation should contain a list of "affected modules" for each data structure so that changes to the structure can be accompanied by appropriate changes to the modules.

EXAMPLES:

GLOSSARY:

Data Structure: Grouping of data elements (variables and constants) into arrays, records, files, etc.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if it is clear that no modules are dependent upon data structure definitions. This will be unlikely in large software programs.

QUESTION DATA SHEET

Question Number D-66

QUESTION: The program has been designed so that functional parts may be easily added or deleted.

CHARACTERISTIC: Expandability (design expandability).

EXPLANATIONS: Programs designed using a top-down, structured methodology often consist of functional parts which are interrelated, yet independent, of one another. That is, each part can be viewed as a "black box" externally. Such parts are usually easily added, deleted, or replaced. However, functional parts designed with complicated, delicate interfaces are more difficult to deal with. An impression should be formed from the module descriptions and program overview information whether the functional parts could be easily added or deleted.

EXAMPLES: Functions executed as a result of a table-driven executive can be easily added and removed by modifying the contents of the table.

GLOSSARY: Functional parts: Primarily modules, but also includes groups of modules that perform major functions.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-67

QUESTION: There is a separate part of the documentation for the description of a program test plan.

CHARACTERISTIC: Instrumentation (format instrumentation).

EXPLANATIONS: Testing is generally regarded as a separate organizational function. It is helpful to those individuals involved in testing to have test information gathered into one part of the documentation.

EXAMPLES: The documentation may include volumes of test information sheets. It may include test plans; acceptance test procedures (ATP), formal or preliminary qualification test (FQT, PQT) procedures. It may include sets of sample input data with expected output data.

GLOSSARY: Program test plan: Set of descriptions and procedures for how the program is to be (or can be, or has been) tested.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if a separate part exists. Answer F if the description does not exist. If for some reason the program test plan description is distributed over several separate parts (e.g., one part per unit/module description), then answer in the range B to E as to the effectiveness of that "separation" from the point of view of program test/retest.

QUESTION DATA SHEET

Question Number D-68

QUESTION: There is a separate part of the documentation for the description of sample test data.

CHARACTERISTIC: Instrumentation (format instrumentation).

EXPLANATIONS: Comparison of input/output data before and after program changes have been made is one of the best ways to assure that changes have been made properly and that no extraneous errors have been introduced.

EXAMPLES:

GLOSSARY: Sample test data: The input and output data used for the program tests.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if a separate part exists. Answer F if the description does not exist. If for some reason the sample test data description is distributed over several separate parts (e.g., one part per unit/module description), then answer in the range B to E as to the effectiveness of that "separation" from the point of view of program test/retest.

QUESTION DATA SHEET

Question Number D-69

QUESTION: There is a separate part of the documentation for the description of program support tools which would aid in testing the program.

CHARACTERISTIC: Instrumentation (format instrumentation).

EXPLANATIONS: Program support tools are not generally a part of the operational software. Descriptions of program support tools are often voluminous and would merely lead to confusion if they were included with descriptions of the operational software. However, the descriptions of program support tools are absolutely necessary and should therefore constitute a separate part of the documentation.

EXAMPLES: A FORTRAN reference manual is an absolute necessity to a scientific programmer, but is definitely not considered to be an integral part of applications software documentation.

GLOSSARY: Program support tools: General debug aids, test/retest software, trace software/hardware features, use of compiler/link editor/library management/configuration management/text editor/display software tools.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if a separate part exists. Answer F if the description does not exist. If for some reason the description of program support tools for testing is distributed over several separate parts then answer in the range B to E as to the effectiveness of that "separation" from the point of view of program test/retest.

QUESTION DATA SHEET

Question Number D-70

QUESTION: A set of test procedures to be used for program check-out are explained.

CHARACTERISTIC: Instrumentation (design instrumentation).

EXPLANATIONS: Program test procedures, in order to be useful, must provide adequate information to completely describe test inputs, outputs, and environment.

EXAMPLES: One good test of the adequacy of the explanation of the program test procedures is for the evaluator to visualize how easy it would be to execute step-by-step one or more of the particular test procedures. If the information is not presented in a step-by-step fashion with a complete discussion of the test environment, test inputs and expected test outputs, then the test will probably be difficult to perform.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: If no test procedures exist, then answer "F."

QUESTION DATA SHEET

Question Number D-71

QUESTION: The set of test procedures provides useful unit testing information.

CHARACTERISTIC: Instrumentation (design instrumentation).

EXPLANATIONS: The test procedures will ordinarily be described in terms of unit testing information and integration testing information. The test procedures should describe test procedures for sub-units of the program as well as for overall program testing. It is often infeasible to test the entire program during modification/testing of only one sub-unit.

EXAMPLES:

GLOSSARY: Unit: Units may be modules, submodules, groups of modules or some other organization depending upon the contractor and the application area.

SPECIAL RESPONSE INSTRUCTIONS: If no test procedures exist, then answer "F."

QUESTION DATA SHEET

Question Number D-72

QUESTION: The set of test procedures provides useful information on limitations/incompleteness.

CHARACTERISTIC: Instrumentation (design instrumentation).

EXPLANATIONS: The testing agency, in order to know what was actually tested and to what extent it was tested, must know the limitations of test procedures.

EXAMPLES: The documentation should contain the ranges of variables tested and not tested as well as modules tested and not tested.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: If no test procedures exist or there is no information on the limitations/incompleteness of the test procedures, then answer F.

QUESTION DATA SHEET

Question Number D-73

QUESTION: The program has been designed with the capability to display test inputs and outputs in summary form.

CHARACTERISTIC: Instrumentation (design instrumentation).

EXPLANATIONS: Many programs process tremendous quantities of data and the test inputs/outputs likewise consist of tremendous quantities of data. In such cases, it is desirable to have a program automatically compare the test data and display only the differences/errors to the maintainer.

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: If the module does not process a great deal of data, so that there is no need to summarize the test inputs/outputs, your answer should lie in the B-E range.

QUESTION DATA SHEET

Question Number D-74

QUESTION: The documentation describes a standardized set of program test data (input and output) that has been designed to exercise the program.

CHARACTERISTIC: Instrumentation (design instrumentation).

EXPLANATIONS: This question relates to both quality and existence of test data. In order to assure that test data properly exercises or tests the program, it must be carefully designed to do so. Randomly assembled data will not usually exercise all parts of the program, whereas carefully designed test data will.

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-75

QUESTION: The documentation indicates that the program has been designed to include software test probes to aid in identifying processing performance.

CHARACTERISTIC: Instrumentation (design instrumentation).

EXPLANATIONS: Test data alone is usually not sufficient to adequately test a program. Certain parts of the program can only be tested by insertion (or activation) of special executable code which is used strictly for testing purposes.

EXAMPLES: If the language provides debug capabilities or such options as conditional compilation, then the designer is much more likely to consider the use of test probes as a normal part of the program. However, it is still possible under more adverse conditions for the design to include separate functions which can be individually invoked for the purpose of collecting appropriate processing performance information.

GLOSSARY:

Include: Presently in-line or can be inserted in-line through activation.

Software test probe: Section of code or special module which collects certain process parameters; generally the activation of the probe can be controlled through user options.

Processing performance: Accuracy, timing, etc.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number D-76

QUESTION: Error checking within the program has been designed to include such features as diagnostic reporting, I/O parameter checking, runtime index range checking, etc.

CHARACTERISTIC: Instrumentation (design instrumentation).

EXPLANATIONS: These particular test tools, as well as many others, are of particular importance to program instrumentaion and test.

EXAMPLES: The documentation describing error processing/error codes/error messages, or perhaps report generation could be checked to determine what type of error checking appears to be done. In addition, the general design conventions/standards might indicate what error checking conventions have been adopted. The source language compiler may have options which allow for the generation of run-time parameter checking (e.g., index range).

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: The evaluator must judge which particular types of instrumentation he feels should be included, and answer A - F according to his estimation of the adequacy of what actually exists.

QUESTIONS DATA SHEET

Question Number D-77

QUESTION: Modularity as reflected in the program documentation contributes to the maintainability of the program.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of modularity to the extent a logical partitioning of software into parts, components, modules has occurred.

EXAMPLES: The software has been partitioned into easily comprehendable "sections." Each "section" is independent from every other "section" as much as is reasonable; i.e., to understand any given "section," requisite knowledge of other "sections" has been kept to a minimum.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general feeling about the modularity of the documentation.

QUESTION DATA SHEET

Question Number D-78

QUESTION: Descriptiveness as reflected in the program documentation contributes to the maintainability of the program.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of descriptiveness to the extent that it contains information regarding its objectives, assumptions, inputs, processing, outputs, components, revision status, etc.

EXAMPLES: Program objectives are explained, subprogram objectives are explained, communication links are either specifically explained or there is a detailed plan for setting up the communication links. Revision status of the documentation is clear. Source listing revision status is either clear or a detailed plan for revision status tracking is explained, etc.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general feeling about the descriptiveness of the documentation.

QUESTION DATA SHEET

Question Number D-79

QUESTION: Consistency as reflected in the program documentation contributes to the maintainability of the program.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of consistency to the extent the software products correlate and contain uniform notation, terminology and symbology.

EXAMPLES: Things are done similarly in different parts of the documentation. Once an individual learns how the documentation is set up, he can turn to any part of the documentation and see exactly what he expects to see. A set of documentation standards appears to have been set up and followed.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general feelings about the consistency of the documantation:

QUESTION DATA SHEET

Question Number D-80

QUESTION: Simplicity as reflected in the program documentation contributes to the maintainability of the program.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of simplicity to the extent that it lacks complexity in organization, language, and implementation techniques and reflects the use of singularity concepts and fundamental structures.

EXAMPLES: The organization of the documentation is logical. Uncomplicated, descriptive terminology is used throughout. Each section or part of the documentation addresses a single subject and is minimally dependent upon other parts for a full understanding.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general impression about the simplicity of the documentation.

QUESTION DATA SHEET

Question Number D-81

QUESTION: Expandability as reflected in the program documentation contributes to the maintainability of the program.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of expandability to the extent that a physical change to information, computational functions, data storage or execution time can be easily accomplished.

EXAMPLES: The documentation contains standards for programming which enhance expandability of the code.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general impression of the overall expandability of the documentation and the program design as reflected in the documentation.

QUESTION DATA SHEET

Question Number D-82

QUESTION: Instrumentation as reflected in the program documentation contributes to the maintainability of the program.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of instrumentation to the extent it contains aids which enhance testing.

EXAMPLES: The documentation contains test cases which show known input and expected output. The documentation also contains a plan or standards for program instrumentation. Some sort of DEBUG mode execution is specifically addressed.

GLOSSARY:
Debug: Removal of bugs.
Bug(s): Latent error(s).

SPECIAL RESPONSE INSTRUCTIONS: Please give your feelings about the instrumentation of the software as reflected in the documentation.

QUESTION DATA SHEET

Question Number D-83

QUESTION: Overall it appears that the characteristics of the program documentation contribute to the maintainability of the program.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software maintainability is a quality of software which is defined as those characteristics which affect the ability of the software engineers to:

1) Correct errors.
2) Add system capabilities through software changes.
3) Delete features.
4) Modify software to be compatible with hardware changes.

EXAMPLES: The program documentation is designed to aid you in maintenance of the subject software. It is not after-the-fact documentation except in those cases where it should be.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general impression as to how well the documentation would aid you in maintenance of the software under study.

C. MODULE SOURCE LISTING QUESTIONS.

Each page within this section corresponds to a question from the Module Source Listing Questionnaire. Many questions have special response instruction which should be reviewed.

Question Data Sheet

Question Number S-1

QUESTION: Functionally related data elements have been organized into logical data structures.

CHARACTERISTIC: Modularity (Data/Control Modularity).

EXPLANATIONS: There may be a physical grouping of functionally related data even though it is somewhat unsatisfactory in showing what the functional relationship is. The evaluator should give a response based upon how easy it is to determine the functional purpose of the data by observing how the data has been organized, grouped, etc.

EXAMPLES: As an example of data structuring, suppose a state vector for an object in track has the information: ID, position, velocity, acceleration. And, suppose a maximum of 100 objects could be in track at one time. The data structuring capabilities of FORTRAN would perhaps represent this situation as:

```
      REAL  POS (100,3),  VEL (100,3), ACC (100,3), STATE (100,10)
      INTEGER ID (100)
      EQUIVALENCE  (STATE (1,1), ID (1)),
     1             (STATE (1,2), POS (1,1)),
     2             (STATE (1,5), VEL (1,1)),
     3             (STATE (1,8), ACC (1,1))
```

GLOSSARY:

Functionally related: Having to do with the same task.

Data elements: Variables, constants.

Data structure: Group of data elements and/or other data structures; e.g., array, record, file, etc.

SPECIAL RESPONSE INSTRUCTIONS. It is not likely that the evaluator's answer will be A or F since most modules will have some logical organization of data, but few will have a superior organization.

Question Data Sheet

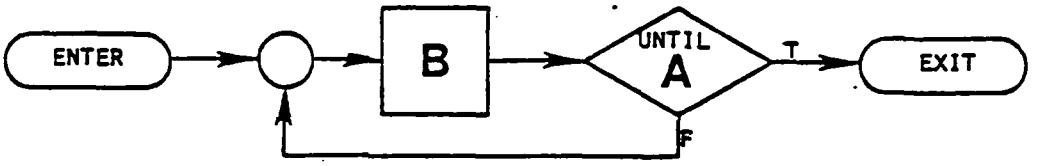
Question Number S-2

QUESTION: The concepts of structured programming have been applied to the control structures in this module.

CHARACTERISTIC: Modularity (Data/Control Modularity).

EXPLANATIONS: The concepts of structured programming control structures can be adopted whether the language is high order or assembly. Some high order language syntax guarantees that only the basic control structures can be used. Other high order languages (e.g., FORTRAN) give some help in constructing the basic control structures with judicious use of the GOTO statement. Assembly languages generally require that a convention be established and MACROS be used in order that only the basic control structures will be consistently used.

EXAMPLES: **DO UNTIL: DO UNTIL condition A, Process B.**



GLOSSARY:

Structured programming control structures: Only three basic control structures are sufficient in structured programming. They are: the SEQUENCE of operation (assignment, add, etc.), IF THEN ELSE (conditional branch to one of two operations and return), and DO WHILE (operation repeated while a condition is true). Only two other constructs should be used: DO UNTIL (operation repeated until a condition is true) and CASE (operation which provides the transfer of program control to a specific location within a compile-time system).

SPECIAL RESPONSE INSTRUCTIONS. The evaluator should form a general opinion as to how well this module has conformed to the use of only these basic control structures. Answer A only if there is no deviation, answer F only if the module's logic structure is extremely disorganized.

Question Data Sheet

Question Number S-3

QUESTION: The use of techniques which involve the sharing of memory locations (e.g., overlay, equivalence, same area) is not excessive.

CHARACTERISTIC: Modularity (Data/Control Modularity).

EXPLANATIONS: Multiple use of memory locations tends to make the program more difficult to maintain.

EXAMPLES: Memory sharing techniques would include dynamic memory management, FORTRAN's EQUIVALENCE operation, COBOL's SAME AREA operation, and memory overlay techniques.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is no sharing of memory locations by this module.

Question Data Sheet

Question Number S-4

QUESTION: The use of global data in this module is not excessive.

CHARACTERISTIC: Modularity (Data/Control Modularity).

EXPLANATIONS: Use of global data tends to decrease the independence of modules and make the program more difficult to maintain.

EXAMPLES: In COBOL, examples of global data would be switch settings and the parameters in a CALL USING statement.

GLOSSARY:
Global data: Variables or constants which can be accessed by more than one module.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is no global data used in this module.

Question Data Sheet

Question Number S-5

QUESTION: The number of entry points of this module is not excessive.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: Multiple entry points tend to increase the number of external interfaces, making program maintenance more complex.

EXAMPLES:

GLOSSARY:
Entry Point: Statement (address) to which control is transferred; first executable statement.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if the number of entry points is one.

Question Data Sheet

Question Number S-6

QUESTION: The number of exit points of this module is not excessive.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: Multiple exit points tend to increase the number of external interfaces, making program maintenance more complex.

EXAMPLES:

GLOSSARY:

Exit point: Statement (address) from which control is transferred; last executable statement; e.g., multiple RETURN statements in a FORTRAN program reflect multiple exit points.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if the number of exit points is one.

Question Data Sheet

Question Number S-7

QUESTION: This module performs only related functional tasks.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: Modules that perform unrelated functional tasks tend to defeat the purpose of modularity and present more information to the maintainer than he needs to perform maintenance in a single functional area.

EXAMPLES: The following list of module task descriptions increases in its complexity and thus would decrease in the score for this question.

1. Module A reads input data, calls Module B.
2. Module B reads input data, conducts error checking, calls Module C.
3. Module C reads input data, conducts error checking, conducts initial processing, calls Module D.
4. Module D calculates the position of the weapon system (navigational tasks) and calculates the amount of fuel remaining.

GLOSSARY:

Functional task: Part, element; a series of contiguous computations, which produce one or more results.

SPECIAL RESPONSE INSTRUCTIONS:

Question Data Sheet

Question Number S-8

QUESTION: Each functional task of this module is an easily recognizable block of code.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: Just as modules should be functionally independent to enhance understandability and modifiability, so should tasks within modules be as independent as possible.

EXAMPLES: Each subsection of code which represents a functional subtask to the primary task of computing a reliability value is easily recognized.

GLOSSARY:
Functional task: Part, element; a series of computations which produce one or more results.

SPECIAL RESPONSE INSTRUCTIONS:

Question Data Sheet

Question Number S-9

QUESTION: It appears that each iteration block within this module has a single entry point.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: Jumps into the body of any iteration block increase coupling to other blocks of code and can defeat iteration block initialization.

EXAMPLES: The following FORTRAN iteration block (implied DO loop) potentially contains more than one entry point:

```
300     CONTINUE
        A = A + B
        I = I + 1
400     CONTINUE
        A = A - B
        J = J + 1
        IF (I.LT.NONE) GOTO 300
        IF (J.LT.NTWO) GOTO 400
```

In COBOL, nested PERFORMs are an example of potentially more than one entry point. (See example under S-10.)

GLOSSARY:

Iteration block: A sequence of instructions in a module that is repeated until a specified set of conditions is either met (DO UNTIL) or not reached (DO WHILE).

Entry point: An instruction at which execution of a block begins.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if all iteration blocks satisfy the one entry point criterion.

Question Number S-10

QUESTION: It appears that each iteration block within this module has a single exit point.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: The intent is that with the exeption of error exits, there should be no jumps out of the body of any iteration block. It is also best (but might only be a minor detraction) if there are no calls to other modules within an iteration block and that the only exit is through the terminal statement which is a natural part of the iteration block. The ESCAPE (jump from the body to the immediately following executable statement) is also only a minor detraction if used properly and not excessively. Multiple exits also tend to increase coupling between blocks of code.

EXAMPLES: COBOL example with three entry and exit points:

```
A-PARA
     ADD 1 TO I.
     MOVE DAT(I) TO OUT(I).
B-PARA
     ADD 1 TO J.
     MOVE DAT(J) TO OUT(I).
EXIT-PARA
     EXIT.
PERFORM A-PARA THRU EXIT-PARA UNTIL I >92.
MOVE 1 TO I.
PERFORM A-PARA UNTIL I >90.
```

GLOSSARY:

Iteration block: A sequence of instructions in a module that is repeated until a specified set of conditions is either met (DO UNTIL) or not reached (DO WHILE).

Exit point: Statement where control leaves the iteration block.

SPECIAL RESPONSE INSTRUCTIONS: The evaluator should answer A only if the exit is always through the natural iteration block exit or if there are no iteration blocks in the module.

Question Data Sheet

Question Number S-11

QUESTION: It appears that each decision block within this module has a single entry point.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: Jumps into the body of any decision block decrease relative independence of blocks of code and can defeat decision block initialization.

EXAMPLES:

GLOSSARY:

Decision block: A sequence of instructions in a module that contains a conditional branch to one of two operations (an IF THEN ELSE control structure).

Entry point: An instruction at which execution of a block can begin.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if all decision blocks satisfy the one entry point criterion or if the program is totally COBOL.

Question Data Sheet

Question Number S-12

QUESTION: It appears that each decision block within this module has a single exit point.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: The intent is that with the exception of error exits, there should be no jumps out of the body of a decision block except as the natural part of the decision syntactic structure. It is also best (but might only be a minor detraction) if there are no calls to other modules within a decision block. The ESCAPE (jump from the body to the immediately following executable statement) is also only a minor detraction if used properly and not excessively. Multiple exits also tend to increase coupling between blocks of code.

EXAMPLES: COBOL decision block with two exit points:

```
IF A = B  GOTO C
   ELSE   GOTO D.
```

GLOSSARY:

Decision block: A sequence of instructions in a module that contains a conditional branch to one of two operations (an IF THEN ELSE control structure).

Exit point: Statement where control leaves the decision block.

SPECIAL RESPONSE INSTRUCTIONS: The evaluator should answer A only if the exit is always through the natural decision block exit.

Question Data Sheet

Question Number S-13

QUESTION: When this module completes execution, control is returned to the calling module.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: Control structure is simplified and maintenance simplified by keeping linkage singular.

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Answer A if there is no calling module (e.g., program module) or this module returns to the calling module. Answer F if this module does not return control to the calling module. Any answer other than A or F should be brought to the attention of the Software Assessment Team chairman.

Question Data Sheet

Question Number S-14

QUESTION: The use of the same variable for both input and output is not excessive in this module.

CHARACTERISTIC: Modularity (Processing Modularity).

EXPLANATIONS: Using variables as both input and output type tends to confuse the maintainer as to their function at specific times during the processing.

EXAMPLES:

GLOSSARY:

Input: Global data or data parameter in argument list used in the module.

Output: Global data or data parameter in argument list and assigned a value within the module.

SPECIAL RESPONSE INSTRUCTIONS:

Question Data Sheet

Question Number S-15

QUESTION: Inputs to this module are described in a preface block.

CHARACTERISTIC: Descriptiveness (Preface Block Descriptiveness).

EXPLANATIONS: Each module should have a preface block which includes a description of each of the inputs. This contributes to overall understanding of the module by the maintainer. Note that in COBOL, the Identification, Environment, and Data Divisions may contain much of the preface block information.

EXAMPLES:

```
C* ARGUMENTS    -INPUT - REAL ARRAYS:
C*                 POS     X,Y,Z POSITIONAL VECTOR (METERS)
C*                 VEL     VELOCITY VECTOR (M/SEC)
C*                 ACC     ACCELERATION VECTOR (M/SEC/SEC)
C*                 STATE   STATE VECTOR (POS, VEL, ACC)
```

GLOSSARY:

Input: Global data or data parameter in argument list used in the module.

Preface block: Separate set of contiguous comments which precedes the main body of the module (a similar block at any other common module location would be satisfactory).

SPECIAL RESPONSE INSTRUCTION: If there is no preface block answer D, E, or F on the basis of whether the module inputs are clearly described at any other place in the module. More than likely the answer should be F if there is no preface block.

If there is a description of inputs in a preface block, then keep in mind that the response anchors A and F are only for extremely good or extremely poor descriptions, respectively.

Question Data Sheet

Question Number S-16

QUESTION: Outputs from this module are described in a preface block.

CHARACTERISTIC: Descriptiveness (Preface Block Descriptiveness).

EXPLANATIONS: Each module should have a preface block which includes a description of each of the outputs from the module. This contributes to overall understanding of the module by the maintainer. See S-15 for COBOL.

EXAMPLES:

```
C*       ARGUMENTS - OUTPUT - INTEGER VARIABLES
C*                   LAT      LATITUDE       (DDDMMSS)
C*                   LONG     LONGITUDE      (DDDMMSS)
C*                   ITIME    CURRENT TIME   (HHMMSS)
C*                   INT      DELAY INTERVAL (HHMMSS)
C*                   ITGT     TIME OVER TGT  (HHMMSS)
```

GLOSSARY:

Output: Global data and data parameters in argument list which are assigned a value within the module.

Preface block: Separate set of contiguous comments which precedes the main body of the module (a similar block at any other common module location would be satisfactory).

SPECIAL RESPONSE INSTRUCTIONS: If there is no preface block answer D, E, or F on the basis of whether the module outputs are clearly described at any other place in the module. More than likely the answer should be F if there is no preface block.

If there is a description of outputs in a preface block, then keep in mind that the response anchors A and F are only for extremely good or extremely poor descriptions, respectively.

Question Data Sheet

Question Number S-17

QUESTION: The purpose of this module is described in a preface block.

CHARACTERISTIC: Descriptiveness (Preface Block Descriptiveness).

EXPLANATIONS: Each module should have within the preface block, a statement of the functions and tasks to be performed by the module. This contributes to overall understanding of the module by the maintainer. See S-15 for COBOL.

EXAMPLES:

```
C*ABSTRACT  : THIS MODULE CALCULATES THE TARGETING
C*            DELAY INTERVAL BY DETERMINING THE
C*            DISTANCE FROM THE CURRENT POSITIONAL
C*            VECTOR TO THE INPUT TARGET LOCATION,
C*            AND DIVIDING BY THE CURRENT VELOCITY.
```

GLOSSARY:

Purpose: Functions, tasks, abstract, summary, etc.

Preface block: Separate set of contiguous comments which precedes the main body of the module (a similar block at any other common module location would be satisfactory).

SPECIAL RESPONSE INSTRUCTIONS: If there is no preface block answer D, E, or F on the basis of whether the module purpose is clearly described at any other place in the module. Since it is quite usual to include embedded comments which clearly describe the module's purpose independent of whether a preface block is present, it would be unusual to respond F to this question statement even if there is no preface block.

If there is a description of the purpose in a preface block, then keep in mind that the response anchors A and F are only for extremely good or extremely poor descriptions, respectively.

Question Data Sheet

Question Number S-18

QUESTION: Modules which call this module are identified in a preface block.

CHARACTERISTIC: Descriptiveness (Preface Block Descriptiveness).

EXPLANATIONS: Each module should have, within the preface block, a list of modules which call it. It may be difficult to verify the completeness of this list, but an attempt should be made using the information available. This contributes to overall understanding of the module by the maintainer. See S-15 for COBOL.

EXAMPLE: C* MODULES CALLING: STAT, INIT1, EXEC

GLOSSARY:

Preface block: Separate set of contiguous comments which precedes the main body of the module (a similar block at any other common module location would be satisfactory).

SPECIAL RESPONSE INSTRUCTIONS: Again, if there is no preface block, answer D, E, or F on the basis of whether the modules which call this module are clearly described at any place in the module.

Question Data Sheet

Question Number S-19

QUESTION: Modules which are called by this module are identified in a preface block.

CHARACTERISTIC: Descriptiveness (Preface Block Descriptiveness).

EXPLANATIONS: Each module should have, within the preface block, a list of modules which it calls. This list can be checked using a scan of the source listing. This contributes to overall understanding of the module by the maintainer. If there are no called modules it should be obvious or so stated. See S-15 for COBOL.

EXAMPLES:

C* MODULES CALLED: FPROB, NAV01, NAV02, ERROUT

GLOSSARY:

Preface block: Separate set of contiguous comments which precedes the main body of the module (a similar block at any other common module location would be satisfactory).

SPECIAL RESPONSE INSTRUCTIONS: It is possible to determine which modules are called by the given module from a cursory look at the source listing. If there is no preface block, then answer D, E, or F on the basis of whether the modules which are called are clearly identified at any other place in the module. If there is a preface block and all the called modules are identified then answer A.

Question Data Sheet

Question Number S-20

QUESTION: Limitations (accuracy, timing, data I/O, etc.) are described as appropriate in a preface block.

CHARACTERISTIC: Descriptiveness (Preface Block Descriptiveness).

EXPLANATIONS: Each module should have, within the preface block, a description of any limitations to its use, such as accuracy and timing. This contributes to overall understanding of the module by the maintainer. See S-15 for COBOL.

EXAMPLES: The following is an example of a description of limitations:

```
C*  LIMITATIONS:  ACCURACY:  LIMITED TO FIVE SIGNIFICANT
C*                DIGITS.
C*                TIMING:    MODULE IS CALLED ONCE EVERY
C*                20 MS, REQUIRES 12 MS TO COMPLETE.
```

GLOSSARY:

Preface block: Separate set of contiguous comments which precedes the main body of the module (a similar block at any other common module location would be satisfactory).

SPECIAL RESPONSE INSTRUCTIONS: If there is no preface block answer D, E, or F on the basis of whether the module limitations are clearly described at any other place in the module. Answer A if there is a preface block and it is either obvious or so stated that there are no limitations which need a description (check the code to verify).

If there is a description of limitations in a preface block, then keep in mind that the response anchors of A and F are only for extremely good or extremely poor descriptions, respectively.

Question Data Sheet

Question Number S-21

QUESTION: Any special processing (e.g., multiple entry/exit, error handling, algorithm peculiarities, etc.) is described in the preface block and understandable.

CHARACTERISTIC: Descriptiveness (Preface Block Descriptiveness).

EXPLANATIONS: Each module should have, within the preface block, a description of any special processing required by the module. This contributes to overall understanding of the module by the maintainer. See S-15 for COBOL.

EXAMPLES:

```
C* ASSUMPTIONS: 1.  FOR INPUT VALUES OF X SUCH THAT
C*                  .0 < ABS (x) < .01, MODULE RETURNS
C*                  X.
C*              2.  FOR INPUT VALUES OF X SUCH THAT
C*                  .01 < ABS (x) < ¶/2, MODULE CALCU-
C*                  LATES THE TRUNCATED SERIES 1/x +
C*                  1/x² + 1/x³.
C*              3.  FOR INPUT VALUES OF X SUCH THAT
C*                  ABS (x) > ¶/2, MODULE RETURNS WITH
C*                  ERROR FLAG SET, ERFL = 1.
```

GLOSSARY:

Preface block: Separate set of contiguous comments which precedes the main body of the module (a similar block at any other common location would be satisfactory if easily locatable).

SPECIAL RESPONSE INSTRUCTIONS: If there is no preface block answer D, E, or F on the basis of whether the module special processing is clearly described at any other place in the module. Answer A if there is a preface block and it is either obvious or so stated that there is no special processing which needs a description (check the code to verify). If there is a description of special processing in a preface block, then keep in mind that the response anchors of A and F are only for extremely good or extremely poor descriptions, respectively.

Question Data Sheet

Question Number S-22

QUESTION: Documentation information (module name, programmer, algorithm references, revision data, etc.) is identified as appropriate in a preface block.

CHARACTERISTIC: Descriptiveness (Preface Block Descriptiveness).

EXPLANATIONS: Each module should contain, within the preface block, general information concerning the development and documentation of the module. Depending upon the application and the program design, the documentation information might also include the component of which the module is a part and the reference sections for written documentation descriptions, flowcharts of the module, etc. See S-15 for COBOL.

EXAMPLES:

```
C*  TITLE                 :   RELIABILITY.
C*  MNEMONIC              :   RELIAB
C*  REVISION DATE         :   07/26/78.
C*  ORIGINAL PROGRAMMER   :   T L PASCHICH, REL DIVISION
C*  REVISION PROGRAMMER   :   N A WEBSTER, SP-48 GROUP
```

GLOSSARY:

Preface block: Separate set of contiguous comments which precedes the main body of the module (a similar block at any other readily locatable area would be satisfactory).

SPECIAL RESPONSE INSTRUCTIONS: If there is no preface block, answer D, E, or F on the basis of whether module documentation information is clearly identified at any other place in the module. If there is no preface block, it is highly unlikely that the documentation information will be identified anywhere else. If there is documentation information identified within a preface block, then keep in mind that the anchor responses A and F are only for extremely complete or extremely incomplete identification, respectively.

Question Data Sheet

Question Number S-23

QUESTION: The comments in this module contain useful information.

CHARACTERISTIC: Descriptiveness (Imbedded Comments Descriptiveness).

EXPLANATIONS: Comments should be used to describe functional blocks of code, complex individual statements, specialized processing, error conditions, limitations, etc.

EXAMPLES: Example of useless comments:

```
C*          CALCULATE ITIME:
              ITIME = ITIME + 3600
C*          CALL SUBROUTINE LATLONG:
              CALL LATLONG (ITIME, VEL, ACC, ERFL)
```

Example of more useful comments:

```
C*          INCREMENT CURRENT TIME BY ONE HOUR:
              ITIME = ITIME + 3600
C*          DETERMINE POSITION AT NEW TIME:
              CALL LATLONG (ITIME, VEL, ACC, ERFL)
```

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: An answer of A or F is not likely. Keep in mind the anchor responses of A and F are for extremely good (best possible) or extremely poor (worst possible) comments.

Question Data Sheet

Question Number S-24

QUESTION: The quantity of comments does not detract from the legibility of the source listings.

CHARACTERISTIC: Descriptiveness (Imbedded Comments Descriptiveness).

EXPLANATIONS: While too few comments may detract from the descriptiveness and understandability of the source listing, too many may also detract. Large modules or modules with complex control structures will suffer in legibility without at least blank comments separating functional blocks or control structures.

EXAMPLES: The following is an acceptable example of the number of comments:

```
C*   COMPUTE SSQ EQUATIONS.
C*
     SST=V2-V1
     SSM=V4-V1
     SSR=V3-V1
     SSWM=V2-V4
     SSE=SSWM-SSR.
C*   COMPUTE DEGREES OF
C*   FREEDOM AND MSQ EQUATIONS.
```

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: An answer of A or F is not likely. Keeping in mind that the anchor responses of A and F are for the best possible quantity of comments or the worse possible quantity of comments, respectively, it is probable that "no comments" deserves an F response, at least for large or complex modules.

Question Data Sheet

Question Number S-25

QUESTION: Transfers of control and destinations are clearly explained.

CHARACTERISTIC: Descriptiveness (Imbedded Comments Descriptiveness).

EXPLANATIONS: It is not necessary that each control branch and destination be commented, but it should be clearly understood what conditions cause a transfer of control and to where control is transferred. The control structure syntax and control parameter name may in itself clearly explain a given transfer of control and its destination without further comment.

EXAMPLES: The following is an acceptable example of explaining transfers of control:

```
C*   ERROR CONDITIONS
C*   IF MSR=0 SET REL = -1.
C*   IF MSE > MSR SET REL = -2.
C*   IF EITHER OF THE ABOVE, SKIP
C*   OVER NORMAL CALCULATION OF REL
     IF (MSR.NE.0) GO TO 1900
     REL = -1.
     GO TO 2000
1900 IF (MSE.LE.MSR) GO TO 2000
     REL = -2.
2000 CONTINUE
```

GLOSSARY:

Transfers of control: Sequential processing and branching performed in the execution within a module. Examples of transfer of control statements in a high level language are: GO TO, CASE, WHILE, and IF THEN ELSE.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if all transfers of control and destinations are clearly explained.

Question Data Sheet

Question Number S-26

QUESTION: Machine-dependencies are clearly commented.

CHARACTERISTIC: Descriptiveness (Imbedded Comments Descriptiveness).

EXPLANATIONS: Often during the life-cycle of a piece of software, the software must be converted to run on an architecturally different or modified processor. To facilitate that conversion, any code that depends on the architecture of the original machine for correct operation should be clearly identified.

EXAMPLES: Code is often dependent on machine word size. If the word size on a specific machine affects the accuracy or precision of a calculation, it should be commented upon.

GLOSSARY:

Machine dependencies: Coding techniques which are unique to the particular computer on which the code is to execute, e.g., certain word size dependencies like FORTRAN I/O formats and shift functions.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if it is clear that there are no machine dependencies in this module. It is unlikely that any module incorporating assembly language code could have a response other than F.

Question Data Sheet

Question Number S-27

QUESTION: Imbedded comments describe each function (block of code) within this module.

CHARACTERISTIC: Descriptiveness (Imbedded Comments Descriptiveness).

EXPLANATIONS: The understandability of module functions is greatly aided by the inclusion of concise, descriptive comments preceding each function. Note that a module's preface block serves as such a descriptive comment for the overall function of the module. If the module performs only one function, then by default this question statement concerns the function description within the preface block (if it exists).

EXAMPLES:

GLOSSARY:

Function: A block of statements which perform some basic computational or logical part of the module's algorithm.

SPECIAL RESPONSE INSTRUCTIONS: It is unlikely that this question statement should get either of the anchor responses A or F.

Question Data Sheet

Question Number S-28

QUESTION: Attributes of each variable used in this module are described by comments and/or source language declarations.

CHARACTERISTIC: Descriptiveness (Imbedded Comments Descriptiveness).

EXPLANATIONS: Typically all global data and module arguments are described in the preface block. Local variables are generally described in a special "declaration" section of the module. Depending upon the source language, the declaration statements may intrinsically describe some or all of a data item's attributes. Description of all attributes of each variable tends to improve maintainability. Variables and parameters with constant values should both be described. Note that in COBOL, the PICTURE phrase contains some of the desired attribute information.

EXAMPLES: The following is an acceptable example of describing variable attributes:

```
C*   REAL VARIABLES
     REAL ALPHA, DFE, DFM, DFR, DFT, ACC, POS, VEL
C*   INTEGER VARIABLES
     INTEGER M, R, INA (M, R)
C* -VARIABLE - DESCRIPTION    - TYPE - UNITS  - RANGE
C*     POS      POSITION VECTOR REAL   METERS  POS > O.
C*     VEL      VELOCITY VECTOR REAL   M/SEC   -500 < VEL
```

GLOSSARY:

Attributes: Type, units, range, description, etc. as appropriate.

Variable: The name or address of data to be used.

SPECIAL RESPONSE INSTRUCTONS: Answer A only if all appropriate attributes for each variable are described in a superior manner. If none of the variables, constants, data used in this module are described, then answer F.

Question Data Sheet

Question Number S-29

QUESTION: Error processing/exits are clearly identified and explained.

CHARACTERISTIC: Descriptiveness (Imbedded Comments Descriptiveness).

EXPLANATIONS:

EXAMPLES: The following identifies an error exit:

```
C*  NEGATIVE VALUES OF REL INDICATE ERROR CONDITIONS.
C*  IF MSR = 0 SET REL = -1
C*  IF MSE > MSR SET REL = -2
C*  IF EITHER OF THE ABOVE, SKIP CALCULATION OF REL.
```

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if all error processing/exits are clearly identified or if there are none to be identified.

Question Data Sheet

Question Number S-30

QUESTION: It appears that a standard for module organization has been followed within this module.

CHARACTERISTIC: Descriptiveness (Implementation Descriptiveness.

EXPLANATION: Application of standards to module organization format insures that the coding will be more readily understandable by anyone referencing the module.

EXAMPLES:

GLOSSARY:

Module organization: Placement of preface block, declarations, error exits, etc. and general format considerations such as imbedded comment format, indentations, etc.

SPECIAL RESPONSE INSTRUCTIONS: Due to the broad nature of this question, a response of A or F is not likely.

Question Data Sheet

Question Number S-31

QUESTION: Variables are declared in a specification/declaration section.

CHARACTERISTIC: Descriptiveness (Implementation Descriptiveness).

EXPLANATIONS: Data whose attributes have been explicitly defined to the compiler are less likely to be used erroneously because the compiler (and hopefully the programmer) can catch any mixed usuages of type, unit, etc.

EXAMPLES: Example of an acceptable declaration of variables:

```
C*  REAL VARIABLES.
    REAL ALPHA, DFE, DFM, UFM, DFT, ACC, POS, VEL
C*  INTEGER VARIABLES
    INTEGER M, R, INA (M, R)
```

GLOSSARY:

Declared: Type (integer, real, array, etc), units, size, etc., is identified.

Specification/declaration section: This may be a formal data declaration section or a section set up by convention for the purpose of declaring variables and other access restrictions, e.g., Data Division in COBOL.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if all variables used in the module are declared, e.g., as in COBOL. (If assembly language is used in the module, it is unlikely that this response would be other than D, E, or F.)

Question Data Sheet

Question Number S-32

QUESTION: Variable names are descriptive of their functional use.

CHARACTERISTIC: Descriptiveness (Implementation Descriptiveness).

EXPLANATION: This question is highly subjective. However, when variables are named descriptively, then the time it takes to understand a module is decreased. The naming conventions which have grown out of particular applications (e.g., avionics) will also influence how functionally descriptive particular names are.

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

Question Data Sheet

Question Number S-33

QUESTION: The module code is indented within control structures to show control flow.

CHARACTERISTIC: Descriptiveness (Implementation Descriptiveness).

EXPLANATIONS: Indentation of control structures enhances the descriptiveness of the module by making the control flow obvious to the maintainer.

EXAMPLES:

```
TEMP = 0
DO 1400 J = 1, R
    DO 1300 T = 1, M
        TEMP = TEMP + INA(1, J1*INA(1, J))
    1300 CONTINUE
1400 CONTINUE
```

GLOSSARY:

Control structure: The basic control structures are SEQUENCE of operations (assignment, add, etc.), decision (IF THEN ELSE - conditional branch to one of two operations and return), and iteration (DO WHILE - operation repeated while a condition is true).

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if all control structures are indented in a manner which clearly shows the control flow or if there are no control structures other than SEQUENTIAL.

Question Data Sheet

Question Number S-34

QUESTION: Statement labels have been named in a manner which facilitates locating a label in the source listing.

CHARACTERISTIC: Descriptiveness (Implementation Descriptiveness).

EXPLANATIONS: The typical method of naming statement labels to facilitate their location is by some kind of ordering. For example, FORTRAN labels are one to five digit numbers and a typical naming scheme is to keep the labels in ascending order from beginning to end of the module. (See example.) Address labels in assembly language are typically alphanumeric names which can be assigned alphabetically from beginning to end of the module.

EXAMPLES: The following is an acceptable example of ascending numerical statement labels:

```
      DO 1600 J = 1,R
      DO 1500 I = 1, M
      TEMP = TEMP * INA (I,J)
1500  CONTINUE
      T3 = T3 * TEMP*TEMP
1600  CONTINUE
      V3 = T3/M
      DO 1800 I = 1, M
      DO 1700 J = 1, M
      TEMP = TEMP* INA (I,J)
1700  CONTINUE
1800  CONTINUE
```

GLOSSARY:

Statement label: The address or location in the source listing which is named; ordinarily the purpose of a statement label is to provide a destination for a transfer of control.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there are no statement labels.

Question Data Sheet

Question Number S-35

QUESTION: The machine cross reference listings appear to be useful.

CHARACTERISTIC: Descriptiveness (Implementation Descriptiveness).

EXPLANATIONS: All labels and symbolic names and constants should appear in a table generated by the assembler/compiler with line number references to show where they are used in the module. Additional entries might call out external references, entry points, and file names.

EXAMPLES:

GLOSSARY:
Machine cross reference listings: The normal compiler/assembler generated cross reference information.

SPECIAL RESPONSE INSTRUCTIONS: Answer F only if no machine cross reference listings are available.

Question Data Sheet

Question Number S-36

NOTE: External consistency questions (S-36 through S-43) require the use of both the source listing and the documentation.

QUESTION: This module's flow chart represents the logic control flow as shown in this module's source listing.

CHARACTERISTIC: Consistency (external consistency).

EXPLANATIONS: Flowcharts are often a great aid to a maintainer who is learning/analyzing the software in order to later make a change. To be of any use, the flowcharts must accurately represent the source code and they must be easy to comprehend. This question is specifically aimed at flow of control.

EXAMPLES:

GLOSSARY:

Flowchart (or equivalent): A logic flow diagram in which symbols are used to represent operations, data, flow, equipment, etc. Examples are: FORTRAN flowchart, Process Design Language (PDL), Hierarchical Input-Processing-Output (HIPO) chart, etc.

Control flow: Logical flow from one control structure to the next control structure.

SPECIAL RESPONSE INSTRUCTIONS: Answer F only if there is no module flowchart or equivalent in the documentation.

Question Data Sheet

Question Number S-37

(See NOTE on S-36)

QUESTION: This module's flow chart represents the data flow as shown in this module's source listing.

CHARACTERISTIC: Consistency (external consistency).

EXPLANATIONS: Flowcharts are often a great aid to a maintainer who is learning/analyzing the software in order to later make a change. To be of any use, the flowcharts must accurately represent the source code and be easy to comprehend. This question is specifically aimed at flow of data.

EXAMPLES:

GLOSSARY:

Data flow: Inputs and outputs to the module as well as any significant intermediate transformations.

SPECIAL RESPONSE INSTRUCTIONS: Answer F only if there is no module flowchart or equivalent in the documentation.

Question Data Sheet

Question Number S-38

(See NOTE on S-36)

QUESTION: The labels in this module's flow chart and the statement labels in this module's source listing are in agreement.

CHARACTERISTIC: Consistency (External Consistency).

EXPLANATIONS: Flowcharts are often a great aid to a maintainer who is learning/analyzing the software in order to later make a change. To be of any use, the flowcharts must accurately represent the source code. This question is specifically aimed at flow chart labelling conventions.

EXAMPLES: In after-the-fact flow charts, all or almost all, of the program labels should appear in the flow charts. In the case where the flow charts were generated before coding, this may not be true.

GLOSSARY:
Label: The address or location which has a name; usually labels are the destination of a transfer of control. The labels in the documentation and source listings should agree in name and sequential location.

SPECIAL RESPONSE INSTRUCTIONS: Answer F only if there is no module flowchart or equivalent in the documentation. Answer A if there are no labels in both the documentation and the source listings. Note: general comments are not considered to be labels.

Question Data Sheet

Question Number S-39

(See NOTE on S-36)

QUESTION: The inputs to this module as described in the documentation correspond to the inputs as shown in this module's source listing.

CHARACTERISTIC: Consistency (External Consistency).

EXPLANATIONS: It is extremely important for each module and its documentation to be in agreement with each other. Most modules interface with other modules, but the maintainer doing interface work usually views other modules mainly through the documentation. This question specifically addresses agreement between input parameters and related documentation.

EXAMPLES:

GLOSSARY:
Inputs: Global data and parameters passed in an argument list which are used within the module.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is a clear one-to-one correspondence between the module's inputs as described in the documentation and as shown in the source listings. If there is no description of the module's inputs in the documentation, then answer F. If the module has no inputs and this is clear from the documentation and the source listings, then answer A.

Question Data Sheet

Question Number S-40

(See NOTE on S-36)

QUESTION: The outputs from this module as described in the documentation correspond to the outputs as shown in the module's source listing.

CHARACTERISTIC: Consistency (External Consistency).

EXPLANATIONS: It is extremely important for each module and its documentation to be in agreement with each other. Most modules interface with other modules, but the maintainer doing interface work usually views other modules only through the documentation. This question specifically addresses agreement between output parameters and related documentation.

EXAMPLES:

GLOSSARY:

Outputs: Global data and parameters passed in an argument list which are assigned a value within the module.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is a clear one-to-one correspondence between the module's outputs as described in the documentation and as shown in the source listings. If there is no description of the module's outputs in the documentation, then answer F.

Question Data Sheet

Question Number S-41

(See NOTE on S-36)

QUESTION: The order of arguments for this module as described in the documentation corresponds to the order of arguments as shown in this module's source listing.

CHARACTERISTIC: Consistency (External Consistency).

EXPLANATIONS: It is extremely important for each module and its documentation to be in agreement with each other. Each module interfaces with other modules, but the maintainer doing interface work usually views other modules only through the documentation. This question specifically addresses agreement between the source listing and the documentation.

EXAMPLES:

GLOSSARY:
Order of arguments: Specific arrangement of information within a global data area, argument list, or other buffer area.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is a clear one-to-one correspondence between the order of arguments as described in the documentation and as shown in the source listings or if it is clear from both documentation and source listings that there are no arguments. If there is no description of the calling sequence in the documentation, then answer F.

Question Data Sheet

Question Number S-42

(See NOTE on S-36)

QUESTION: The module processing as described in the documentation corresponds to the implemented processing as shown in this module's source listing.

CHARACTERISTIC: Consistency (External Consistency).

EXPLANATIONS: It is extremely important for each module and its documentation to be in agreement with each other. Each module will occasionally have new personnel assigned to maintain it. Documentation is a necessary learning tool. Furthermore, other personnel who desire to know how data is transformed within the module rely heavily upon the documentation as an accurate representation of the implemented processing.

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Answer F if the module processing is not described in the documentation.

Question Data Sheet

Question Number S-43

(See NOTE on S-36)

QUESTION: The programming conventions established in the documentation for source code development have been followed within this module.

CHARACTERISTIC: Consistency (External Consistency).

EXPLANATIONS: Adherence to design standards insures that the coding will be more readily understandable by anyone involved in the project (who presumably knows the standards). It also helps to insure that auditors, code checkers, and testing agencies have done (will do) their jobs properly.

EXAMPLES:

GLOSSARY:

Programming conventions: Preface content, variable/module names, source code and imbedded comment formats, I/O, error handling, etc.

SPECIAL RESPONSE INSTRUCTIONS: Answer F if it is clear that no programming conventions have been established.

Question Data Sheet

Question Number S-44

QUESTION: The delineation of comments is uniform within sections of this module.

CHARACTERISTIC: Consistency (Internal Consistency).

EXPLANATIONS: The concept is that conventions should be established which more or less give a template for source format. Adherence to the convention can be either manual or automated. This gives a uniformity to the body of the code as well as to the placement, use of, and delineation of comments which clarify the purpose of the code. It is very distracting from a consistency viewpoint to have comments delineated in a variety of ways; e.g., some have spaces, some have asterisks, some comments are imbedded, etc.

EXAMPLES:

GLOSSARY:
Delineation: Method of separating; highlighting, placement, etc.

SPECIAL RESPONSE INSTRUCTIONS:

Question Data Sheet

Question Number S-45

QUESTION: Each variable in this module is considered to be of one (and only one) data type for all occurrences.

CHARACTERISTIC: Consistency (Internal Consistency).

EXPLANATIONS: Consistent programming practices lead to more maintainable code. Use of a storage location for more than one data type can lead to confusion as to what is contained in the location, how to interpret it, how to process it, etc.

EXAMPLES: 1. The use of a variable defined as integer in bit manipulation operations is an example of a variable being used as a type (bit string) other than its defined type.

2. A variable described as an alphanumeric character (PIC XX) should not be used both as an integer (e.g., in an addition or as a counter) and as an alpha character.

GLOSSARY:

Data type: A set of attributes used to define a class of data; e.g., INTEGER, REAL, BOOLEAN, CHARACTER, ARRAY OF INTEGER, etc.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if each variable has only one associated data type.

Question Number S-46

QUESTION: Each variable in this module has only one function.

CHARACTERISTIC: Consistency (Internal Consistency).

EXPLANATIONS: Multiple use of a storage location can lead to confusion.

EXAMPLES: Temporary local variables are frequently used for more than one purpose, in order to conserve storage; or the use of a variable may depend on certain control parameters, hence giving rise to multiple uses; or storage allocated in assembly and referenced by an address name (variable) may be partitioned and used in a variety of ways.

In COBOL, consider RENAME areas and variables in any Data Division.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if each variable used in the module has only one functional use.

Question Data Sheet

Question Number S-47

QUESTION: Global variables are distinguishable from local variables by a naming convention.

CHARACTERISTIC: Consistency (Internal Consistency).

EXPLANATIONS: Uniform adherence to global variable naming standards helps the maintainer to easily discriminate between local and non-local variables. The non-local variables require extra care due to interface requirements.

EXAMPLES: Each global variable has a six character name, each local variable has a less-than-six character name; or, all global variables begin with the letter G, all local variables begin with the letter L.

GLOSSARY:
Global variables: Variables that can be accessed by more than one module.

SPECIAL RESPONSE INSTRUCTIONS: Answer F if there is no naming convention for global and local variables. Even if there is a naming convention, however, it may not be one which allows for clear distinction between global and local variables (i.e., the answer may not be A). If there are no global variables, answer A.

Question Data Sheet

Question Number S-48

QUESTION: The use of indentation is uniform within this module.

CHARACTERISTIC: Consistency (Internal Consistency).

EXPLANATIONS: Indentation can be helpful in identifying control structures, highlighting peculiar code, etc. However, for indentation to be useful, it must be used in the same way throughout the code.

GLOSSARY:
Indentation: The use of non-essential blank characters in any source line statement (usually associated with the first part of a line with executable statements).

SPECIAL RESPONSE INSTRUCTIONS:

Question Data Sheet

Question Number S-49

QUESTION: The information in the preface block is consistent with the associated source code.

CHARACTERISTIC: Consistency (Internal Consistency).

EXPLANATIONS: Preface blocks are extremely helpful to the maintainer in understanding the code, but to be helpful, they must be accurate.

EXAMPLES: Inputs explained agree with inputs as coded.
Outputs explained agree with outputs as coded.
Processes explained agree with processes as coded.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: If there is no preface block, then answer "F".

QUESTION DATA SHEET

Question Number S-50

QUESTION: The source language for this module is a high order language (HOL).

CHARACTERISTIC: Simplicity (General coding simplicity).

EXPLANTATION:

EXAMPLES:

GLOSSARY:

High Order Language: A programming language that does not reflect the structure of any one given computer or any class of computers; other than assembly, machine, micro code, etc.; e.g., FORTRAN, PL/1, Pascal, Ada, JOVIAL,.....

SPECIAL RESPONSE INSTRUCTIONS:

Answer A if the source language is 100% HOL.
Answer F if the source language is 0% HOL.
Answer B-E otherwise.

QUESTION DATA SHEET

Question Number S-51

QUESTION: The control flow of this module is essentially from top to bottom.

CHARACTERISTIC: Simplicity (General Coding Simplicity)

EXPLANATIONS: Logical flow which is not top to bottom is usually because of the use of unstructured control (e.g., unconditional branches and other forms of GOTO's) paths. Use of an iteration control structure is not considered to be a deviation from top to bottom control flow.

EXAMPLES: The following code segment illustrates a top to bottom control flow.

```
      DO 20 I = 1,10
         IF (Q) GO TO 10
         B = I
         GO TO 20
10       CONTINUE
         I = I + 1
20    CONTINUE
```

The following code segment illustrates a deviation from top to bottom control flow (as well as other bad coding practices).

```
5     CONTINUE
      C = C + B
      DO 10 I = 1,10
      B = I
      IF (Q) GO TO 5
      I = I + 1
10    CONTINUE
```

GLOSSARY:

Control flow: Logical flow from one control structure to the next control structure.

Control structure: The basic control structures are SEQUENCE of operations (assignment, add, etc.), decision (IF THEN ELSE - conditional branch), and iteration (DO WHILE - operation repeated while a condition is true).

SPECIAL RESPONSE INSTRUCTIONS:

It can be very time consuming to scan source listings and clearly determine that the control flow is top to bottom (in fact, control structures may not be easy to determine). Hence, it is unlikely to expect an answer of A.

Likewise, control does eventually get from top to bottom in most modules and usually can't be considered to be in the worst-possible-case class of examples (answer of F).

QUESTION DATA SHEET

Question Number S-52

QUESTION: This module contains very little extraneous code.

CHARACTERISTIC: Simplicity (General Coding Simplicity)

EXPLANATIONS: Coding that serves no use causes confusion and makes the program logic more difficult to understand.

EXAMPLES:

GLOSSARY:
Extraneous code: Dead code or blatantly inefficient code.

SPECIAL RESPONSE INSTRUCTIONS:
Answer A only if there is no extraneous code.

QUESTION DATA SHEET

Question Number S-53

QUESTION: There is minimal use of specialized coding techniques is this module.

CHARACTERISTIC: Simplicity (General coding simplicity).

EXPLANATIONS: Specialized coding techniques require a more in-depth knowledge of the internal workings of the machine and language, and therefore increase the difficulty of maintaining the system.

EXAMPLES:

GLOSSARY:

Specialized coding: Masking, bit manipulation, machine dependencies, imbedded assembly code, etc.

SPECIAL RESPONSE INSTRUCTIONS

Answer A only if there are no specialized coding techniques used in the module. For assembly language programming it is unlikely that the response would be other than F.

QUESTION DATA SHEET

Question Number S-54

QUESTION: Esoteric (clever) programming is avoided in this module.

CHARACTERISTIC: Simplicity (General coding simplicity).

EXPLANATIONS: Esoteric programming techiques increase the workload on the maintainers by requiring more time to try and figure out what the originator of the code was trying to do.

EXAMPLES: The following example of clever FORTRAN programming is illustrative of an unfortunately large set of possible examples.

```
      DO 10 I=1, N
      DO 10 J=1, N
10    A(I,J) = (I/J)*(J/I)
```

In this example a matrix is being initialized. However, when I>J then J/I = 0 due to intrinsic FORTRAN integer division rules and likewise, when J>I then I/J = 0. The net result is that all diagonal elements, A(I,I), of the matrix are assigned the value 1.0 and all other elements are assigned the value 0.0 Clever, but a more understandable and more efficient version is given below.

```
      DO 20 I=1, N
           DO 10 J=1, N
10         A(I,J) = 0.0
20    A(I,I) = 1.0
```

GLOSSARY:

Esoteric: Overly clever or tricky.

SPECIAL RESPONSE INSTRUCTION: It is unlikely that the evaluator would be able to determine except through a detailed study of the source listings whether clever programming techniques were completely avoided or whether the module deserves to rank with the worst in this respect. Hence, responses at the anchor points A or F are fairly unlikely.

QUESTION DATA SHEET

Question Number S-55

QUESTION: GO TO-like branch statements in this module are used only where essential.

CHARACTERISTIC: Simplicity (General coding simplicity).

EXPLANATIONS: GO TO statements increase the number of labeled statements (which may be in another portion of the program) thereby increasing the difficulty of following the program logic. They tend to defeat the purpose of structured programming.

EXAMPLES: The use of a computed GO TO to construct a Case construct and the use of a GO TO as an escape are illustrated below. Similar constructs can be generated in assembly language.

Case:

```
      IF ((M.LT.1) .OR. (M.GT.4)) GO TO 50
      GO TO (10,20,30,40), M
10    A = A+1.
      GO TO 50
20    B = B+1.
      GO TO 50
30    C = C+1.
      GO TO 50
40    D = D+1.
50    CONTINUE
```

Escape:

```
      DO 20 I = 1,1000
           DO 10 J = 1,1000
                A = A+1.
10         IF (A) GO TO 30
20    CONTINUE
30    CONTINUE
```

GLOSSARY:

GO TO-like branch statements: Control structures with conditional or unconditional branching.

SPECIAL RESPONSE INSTRUCTIONS:

Answer A only if there are no GO TO like branch statements.

QUESTION DATA SHEET

Question Numbers S-56

QUESTION: There is reasonable use of statement labels in this module.

CHARACTERISTIC: Simplicity (General coding simplicity).

EXPLANATIONS: Statement labels referenced may or may not be in the section of code under consideration by the maintenance programmer and, if not, increase the workload on the maintainer by requiring "hunting". Reasonable use of statement labels will allow a maintenance programmer to follow control flow with minimum effort.

EXAMPLES:

GLOSSARY:

Statement label: Address or location to which control can be transferred within a module.

Use: Number of statement labels as well as the number of times the labels serve as the destination of a transfer of control.

SPECIAL RESPONSE INSTRUCTIONS:

If there are no statement labels, then answer A.

QUESTION DATA SHEET

Question Number S-57

QUESTION: A knowledge of mathematics beyond basic algebra is not required to understand the mathematical functions performed by this module.

CHARACTERISTIC: Simplicity (General coding simplicity).

EXPLANATIONS: Higher level mathematical functions require the maintainer to be educated to that level and increase the training levels required of the programmers.

EXAMPLES:

GLOSSARY:

Basic algebra: Functions (including trigonometric and exponential functions), equations, polynomials (including series), graphing of functions, basic manipulations, etc.; excludes calculus, Fourier transforms, statistical algorithms, etc.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number S-58

QUESTION: This module contains a minimal number of compound data structures.

CHARACTERISTIC: Simplicity (Singular coding simplicity)

EXPLANATIONS: Data structures are either primitive data types (e.g., INTEGER, REAL, BOOLEAN, CHAR, etc.) or are structures with components. A component's type may be primitive or may itself be a structure. The concept is that reasonable use of a small number of primitive structures is simple and that deviations (whether because of necessity or other reasons) such as the use of the nested or compound structures tends to reflect more complexity. Of course, this concept can be carried to extremes. If the software developer has done a reasonable design, then the type of data structures used should reflect the functional use of the data and hence the complexity of the programming task.

EXAMPLES: Compound structures could be arrays, arrays of arrays, records, FORTRAN COMMON (with more than one primitive component), etc. In assembly language the use of a block of storage as an array (e.g., via manual index computations) would constitute a compound data structure. However, for the most part, assembly language modules will show a response closer to A.

Note that the use of fields (e.g., bit strings) within a computer word is considered to be a compound data structure.

GLOSSARY:

Compound: nested or non-singular; any non-primitive data type.

Data Structure: Group of data elements; e.g., array, record, file, etc.

SPECIAL RESEPONSE INSTRUCTIONS: Consider A only if there is no use of compound data structures.

QUESTION DATA SHEET

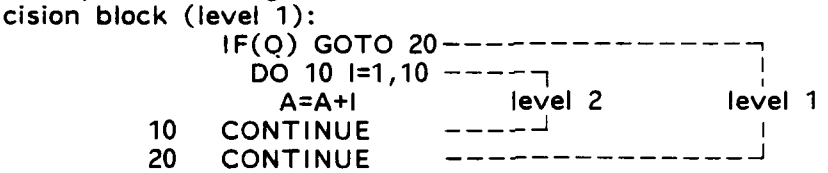
Question Number S-59

QUESTION: This module contains a minimal number of compound control structures.

CHARACTERISTIC: Simplicity (singular coding simplicity).

EXPLANATIONS: The fewer number of compound control structures a module has, the easier the logic is to understand and maintain.

EXAMPLES: The following illustrates one compound structure, namely the nesting of an iteration block (level 2) within a decision block (level 1):



In COBOL, examples are nested PERFORMs and nested IFs.

GLOSSARY:

Compound: Contains more than one of the primitive control structures (e.g., SEQUENCE, DECISION, or ITERATION), often by nesting or imbedding one control structure within another.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there are no compound control structures. An answer of F should reflect an extremely poor control structure organization. It is not possible to specify precise guidelines for the anchor F response. However, the greater number of nested control structures and/or the greater the level of nesting for any individual nest should be reflected through a lower response (A highest, F lowest).

QUESTION DATA SHEET

Question Number S-60

QUESTION: Each physical source line in this module contains at most one executable source statement.

CHARACTERISTIC: Simplicity (singular coding simplicity).

EXPLANATIONS: Source lines containing more than one executable statement increases the probability of missing or wrongly interpreting the program logic.

EXAMPLES: In COBOL: MOVE X TO Y COMPUTE X=Y+1. (bad)
MOVE X TO Y
COMPUTE X = Y+1. (good)

GLOSSARY:
Executable statement: As defined by the language; for our purposes this excludes comments, data declarations, and variable/constant declarations.

SPECIAL RESPONSE INSTRUCTIONS:
The response should be based on a relative average of the number of source lines which contain only one executable statement to the total number of source lines with at least one executable statement.

A - 0% no source line has more than one executable statement.
B - 0% to 10%
C - 10% to 40%
D - 40% to 60%
E - 60% to 80%
F - Almost every source line has more than one executable statement.

QUESTION DATA SHEET

Question Number S-61

QUESTION: There is minimal use of compound Boolean expressions in this module.

CHARACTERISTIC: Simplicity (singlular coding simplicity).

EXPLANATIONS: Compound Boolean expressions are primarily used in high order languages where Boolean operators such as AND and OR are available.

EXAMPLES: Examples of compound Boolean expressions are illustrated below:

```
(A.AND.B).OR.(C.AND.D)
(A.AND.B.AND.C)
```

A typical use might be as follows:

```
IF(I.GT.1).AND.(J.LT.10.OR.J.GE.K)) GOTO 20
```

GLOSSARY:

Boolean: Logical combinatorial system that symbolically represents relationships between sets or propositions; usually using the operators AND, OR and NOT.

Compound Boolean expressions: Linking or nesting involving more than one Boolean operator.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there are no compound Boolean expressions.

QUESTION DATA SHEET

Question Number S-62

QUESTION: The numoer of expressions used to control branching in this module is manageable.

CHARACTERISTIC: Simplicity (size simplicity).

EXPLANATIONS: The count of control expressions is closely related to the number of independent cycles in a module. The more control expressions there are, the more complex the control logic tends to be. An actual count of the number of control expressions in the module may be obtained as shown in the example.

EXAMPLES: The following examples indicate how to count the control expressions:

CONTROL STRUCTURE	STATEMENT	CONTROL EXPRESSION	COUNT
Decision	IF (A.OR.B) GOTO 10	A;B	2
	IF (A.AND.B) GOTO 10	A;B	2
	IF (C.GT.D) GOTO 10	C.GT.D	1
	IF (A.AND.B).OR.(C.GT.D)) GOTO 10	A;B;C.GT.D	3
	CASE (I) OF 1: A 2: B 3: C END CASE	(alternatives)	(number of alternatives less one)
Iteration	DO 10 I=1,10	I.LT.1	2
	IF (A.GT.I) GOTO 20	I.GT.10	
	10 CONTINUE	A.GT.I	10 (repetition)
	20 CONTINUE		

GLOSSARY:

Control expression: IF, CASE, or other decision control expression. DO, DO-WHILE, or other iterative control expression.

SPECIAL RESPONSE INSRUCTIONS: The following guidelines will anchor the A and F responses, but are fairly subjective (especially the F anchor). The guidelines for the A response is suggested from other independent research. Remember to count all repetitions of the same control expression also.

Answer A if count $\leq$ 10.

Answer F if count $>$ 50.

QUESTION DATA SHEET

Question Number S-63

QUESTION: The number of unique operators in this module is manageable.

CHARACTERISTIC: Simplicity (size simplicity).

EXPLANATIONS: The concept is that the more distinct or unique operators there are, the more discriminations one must make in order to understand the module's function. Again, a count is to be made, but keep in mind that it is the total number of unique operators which are to be counted, not the total number of operators (which would include the repetitious use of each operator). It is not intended that the evaluator spend a great amount of time counting operators. In a period of a few minutes using the guidelines below, it should be possible to obtain a reasonable estimate of the number of unique operators.

EXAMPLE: The minimal number of operators for any algorithm is 2 (one operator is the transfer of control to the algorithm being invoked and the other is an assignment or grouping symbol for transfer of the result; e.g., Y = RANDOM or CALL RANDOM (Y)).

GLOSSARY: The following are guidelines as to what constitutes an operator. Some examples are also provided. The list below should be considered representative and not complete:

1. All typical language verbs are operators;
   e.g., unary op: negation (-), set complement (')
   binary op: addition (+) subtraction (-)
   mulitplication (*) division (/)
   exponentiation (**) assignment (=)
   relation op: less than (LT, <)
   greater than (GT, >)
   equal (EQ, =)
   not equal (NE, =)
   less than or equal (LE, ≤)
   greater than or equal (GE, ≥)
   logical op: AND
   OR
   NOT
   control op: decision (IF THEN ELSE, FORTRAN logical/arithmetic, assembly, branch, CASE)
   iterative (DO loop, DOWHILE, DOUNTIL, etc.)
   sequential grouping (BEGINEND, DO)

GOTO (each distinct GOTO label is a unique operator)

2. There are various groupings of terms, each of which is an operator.

grouping: expression
argument list

3. Each reference call is an operator.

external: module/function call (each unique one is counted).
intrinsic function call (MOD, MAX, ABS, SHILFT, etc.)
external function call (SIN, COSINE, LOG, EXP, SQR, etc.)

SPECIAL RESPONSE INSTRUCTIONS: The anchors below for A and F responses are not sacred, but are reasonable.

Answer A if count $\leq$ 10.
Answer F if count $\geq$ 50.

QUESTION DATA SHEET

Question Number S-64

QUESTION: The number of unique operands in this module is manageable.

CHARACTERISTIC: Simplicity (size simplicity)

EXPLANATIONS: The concept is that the more operands there are, the more discriminations one must make in order to understand the module's function. It is not intended that the evaluator spend a great amount of time counting operands. In a period of a few minutes it should be possible to obtain a reasonable estimate of the number of unique operands.

EXAMPLES: An array (name) is considered a single operand.

GLOSSARY:
Operands: Variables and constants.

SPECIAL RESPONSE INSTRUCTIONS: The anchors below for A and F responses are not sacred, but are reasonable.
Answer A if count $\leq$ 40.
Answer F if count $\geq$ 240.

QUESTION DATA SHEET

Question Number S-65

QUESTION: The number of executable statements in this module is manageable.

CHARACTERISTIC: Simplicity (size simplicity).

EXPLANATIONS: The fewer lines of code in a particular module, the easier it is to understand and maintain.

EXAMPLES:

GLOSSARY:

Executable statements: As defined by the language; for our purposes this excludes comments, data declarations, and variable/constant declarations.

SPECIAL RESPONSE INSTRUCTIONS: The following guidelines for anchor responses are not sacred, but are reasonable.

Answer A if count $\leq$ 50.

Answer F if count $\geq$ 300.

QUESTION DATA SHEET

Question Number S-66

QUESTION: There is a minimal mixing of I/O functions and other application functions in this module.

CHARACTERISTIC: Expandability (general expandability).

EXPLANATIONS: The concept is that mixing of I/O code and other operational computations makes it difficult to modify either the I/O or the operational computations which probably are bound to the I/O in some manner. By keeping the mix to a minimum it is easier to maintain.

EXAMPLES: Separate modules that perform output functions for the program.

GLOSSARY:

Functions: Tasks performed by submodules, groups of related statements, etc., as appropriate.

I/O functions: Actual external device interfaces; e.g., FORTRAN read and write, operating system file management calls, hardware controller interface calls, etc.

Application functions: Any functions which provide specific operational computations.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is no mix (i.e., the functions are either entirely I/O or entirely non-I/O).

QUESTION DATA SHEET

Question Number S-67

QUESTION: There is minimal mixing of machine dependent functions and other application functions in this module.

CHARACTERISTIC: Expandability (general expandability).

EXPLANATIONS: The concept is that mixing of machine dependent functions with other operational functions makes it difficult to modify either the machine dependent code or the operational computations.

EXAMPLES: In-line assembly code, special instructions to address extended memory, etc. within the module.

GLOSSARY:

Functions: Tasks performed by submodules, groups of related statements, etc., as appropriate.

Machine dependent functions: Functions which are particular to the host computer; e.g., architectural peculiarities.

Application functions: Any functions which provide specific operational computations.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is no mix (i.e, the functions of this module are either entirely machine-dependent or non-machine-dependent).

QUESTION DATA SHEET

Question Number S-68

QUESTION: Constants used more than once in this module are parameterized.

CHARACTERISTIC: Expandability (general expandability).

EXPLANATIONS: It is easier and less error-prone to change a constant in one place rather than every place that it is used.

EXAMPLES:

GLOSSARY:
Parameterized: Referenced by name, not by the actual constant value; e.g., PI instead of 3.1415926.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if all constants used more than once have been parameterized, or if no such conditions occur.

QUESTION DATA SHEET

Question Number S-69

QUESTION: There is minimal use of processing-dependent code (e.g., relative addressing, self-modifying code, etc.) in this module.

CHARACTERISTIC: Expandability (general expandability).

EXPLANATIONS: The concept is that code should not have built-in processing dependencies which make code modification difficult. These processing dependencies may be internal module dependencies, may be code which generates dependencies in other modules, or may be code which depends upon some external processing state for its functional effect.

EXAMPLES: Example of processing dependence is a module whose effect is dependent upon the number of times it has been executed. Frequently a module performs only certain processing functions the first time or the Nth time it is executed. Another example would be relative addressing, as a "jmp + 5" instruction which might catch a maintenance programmer making changes within those next five instructions.

GLOSSARY:

Self-modifying code: Code which changes the actual code instructions of some other part of the module or of some other module depending upon the current processing state; dynamic code generation.

Relative addressing: Code designed to use the location of the current instruction as a basis for referencing other locations.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is no processing-dependent code.

QUESTION DATA SHEET

Question Number S-70

QUESTION: The size of any data structure which affects the processing logic of this module is parameterized.

CHARACTERISTIC: Expandability (processing expandability).

EXPLANATIONS: Since it may be necessary to modify the size of a given data structure, it is very helpful not to have to be concerned with catching all the implicit references to the structure size within the algorithm code which uses the data structure.

EXAMPLES: Some examples would be the use of an array size to control the number of iterations, or to control indexing into the array or other storage locations.

GLOSSARY:

Parameterized: Referenced by name, not by actual constant value.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is no use of data structure size information (implicit or explicit) within the processing logic of the module except where the size has been parameterized.

QUESTION DATA SHEET

Question Number S-71

QUESTION: Any constants (e.g., accuracy, convergence, timing) which affect processing in this module are parameterized.

CHARACTERISTIC: Expandability (processing expandability).

EXPLANATIONS: The concept is that any metric constant (or specific requirement) which in some manner controls the processing logic of the module should not be used as a literal constant, but should be parameterized to allow for modification ease should the necessity arise.

EXAMPLES:

GLOSSARY:

Parameterized: Referenced by name, not by actual constant value.

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if all constants which affect processing are parameterized either locally or globally, or if there are no such constants.

QUESTION DATA SHEET

Question Number S-72

QUESTION: The contribution of this module to the consumption of frame time can be determined.

CHARACTERISTIC: Expandability (processing expandability).

EXPLANATIONS: In many applications, timing considerations are of paramount importance. Time consumption is often the single constraint on processing expandability. Therefore, the module should contain some mechanism with which time consumption can be measured.

EXAMPLES: Conditional compilation statements which are used for determination of time consumption may be included in the code. Breakpoints used for determination of time consumption may have been included in the code. Comments may describe present execution time and/or how to determine execution time should the module be modified.

GLOSSARY:
Frame: a predetermined period of time in which the system must perform specific functions.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if there are no critical time constraints in the system (e.g., non-real-time). If the contribution can be estimated, rather than determined, accurately enough to satisfy the requirements of system software maintenance, answer A.

QUESTION DATA SHEET

Question Number S-73

QUESTION: The volume of data which this module can process does not appear to be limited.

CHARACTERISTIC: Expandability (processing expandability).

EXPLANATIONS: The concept is that it would be best if a module does not have to be modified simply because more data is to be processed.

EXAMPLES: 1. A sort algorithm may well be volume bound or may not be, depending upon how the processing algorithm is set up to handle the input and output of the data.

2. A read until end-of-file into an array implies limitation.

3. An example of an unlimited process would be a program that produces an 80-80 listing.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number S-74

QUESTION: It appears that functional parts could be easily inserted, deleted, or replaced within this module.

CHARACTERISTIC: Expandability (proessing expandability).

EXPLANATIONS: The concept is that the module or its functional parts should be modifiable on a functional insert/delete/replace basis. The functional parts of a module should be designed such that their strengths are maximized and their interconnectedness is minimized.

EXAMPLES: A trigonometric function is easily replaceable. Numerical labels in the module might be incremented by 10 or even 100 to allow for insertions at a later date.

GLOSSARY:

Functional part: Submodule, task, contiguous statements performing a basic computational step.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number S-75

QUESTION: This module contains checks for possible out-of-bound array subscripts.

CHARACTERISTICS: Instrumentation (processing instrumentation).

EXPLANATIONS: Miscalculating values for variables used as array subscripts is a source of error during array processing. Values that are negative or that exceed the declared dimension of an array will cause unpredictable results. The program, either through the compiler or through actual programming consideration, should contain adequate tests wherever variables are used as subscripts to insure that the variables have reasonable values.

EXAMPLES:

GLOSSARY:

Array: Either an explicitly declared array or a storage area (e.g., in assembly) which is effectively used as an array.

SPECIAL RESPONSE INSTRUCTIONS: If the module does not use arrays or some other type of indexing which could exceed the actual maximum size, then answer A. Also answer A if it is clear that none of the array subscripts can go out of bounds.

QUESTION DATA SHEET

Question Number S-76

QUESTION: This module contains checks to detect possible undefined operations.

CHARACTERISTIC: Instrumentation (processing instrumentation).

EXPLANATIONS:

EXAMPLES: Divide by zero check:

```
IF (A.NE.O)
    THEN X = B/(2*A)
```

GLOSSARY:

Undefined operation: Divide by zero, square root of negative number, singular matrix operation, numerical divergence, etc.

SPECIAL RESPONSE INSTRUCTIONS: If the module does not contain any operations which could give an undefined result, then answer A.

QUESTION DATA SHEET

Question Number S-77

QUESTION: This module contains a minimal amount of code which would require lower-level detailed testing.

CHARACTERISTIC: Instrumentation (processing instrumentation).

EXPLANATIONS: The concept is that a module that contains a minimal number of complex functional parts (e.g., numerical algorithms, interface coordination, timing schemes, etc.) is easier to maintain because less lower-level testing is required.

EXAMPLES:

A module which contains its own trigonometric function algorithm, rather than using the operating system's library routine, would require additional testing to insure that the algorithm is (and remains) operationally sound.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Answer A only if there is no code in the module which would require detailed testing.

QUESTION DATA SHEET

Question Number S-78

QUESTION: Source listing comments suggest or reference input data and associated output results for use in testing this module.

CHARACTERISTIC: Instrumentation (control of instrumentation).

EXPLANATIONS:

EXAMPLES:

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS:
If there are no procedures (let alone comments for procedures) for testing this module, answer F.

QUESTION DATA SHEET

Question Number S-79

QUESTION: Diagnostic messages/error codes are output when an illegal input to this module is encountered.

CHARACTERISTIC: Instrumentation (control of instrumentation).

EXPLANATIONS: The basic question is whether all input data is checked as appropriate for format, range, or any other attribute which might make the data invalid. It is especially important to validate input data which is a logic control parameter (e.g., decision parameter, loop index). The validation process should include the appropriate diagnostic messages and/or error codes.

EXAMPLES:

GLOSSARY:

Input: Includes data passed to the module both at entry to the module and from externally referenced modules.

SPECIAL RESPONSE INSTRUCTIONS: It may be that all data which is input to this module either does not require validation or is validated in some other (perhaps one module for that purpose) place; if it is clear from the comments in the source listings that this is the case, then answer A.

QUESTION DATA SHEET

Question Number S-80

QUESTION: Diagnostic messages/error codes are output wherever an internal module failure could occur.

CHARACTERISTIC: Instrumentation (control of instrumentation).

EXPLANATIONS: Internal module failure consists of algorithm failures due to error conditions which are recognized within the module; these error conditions may be due to out-of-bound array subscript, undefined operations, algorithm inadequacy, etc. This failure is not due to error conditions from input data checks, but may be based upon perfectly valid input data.

EXAMPLES:
```
IF ((A*A - B*B + C*C).EQ.0.0) GOTO 200
XALRT = (A*A + B*B)/(A*A - B*B + C*C)
IF (OVRFL) GOTO 99
200 CONTINUE
```
(In the above example, a standard divide by zero check has been performed; however, internal module failure could still occur if an indefinite value of XALRT were to be used - thus the overflow check.)

GLOSSARY:

Algorithm: A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.

Internal Module Failure: Unacceptable interruption of processing within the module.

SPECIAL RESPONSE INSTRUCTIONS: Answer A if no internal module failures are possible.

QUESTION DATA SHEET

Question Number S-81

QUESTION: Intermediate results within this module can be selectively collected for display.

CHARACTERISTIC: Instrumentation (control of instrumentation).

EXPLANATIONS:

EXAMPLES:

GLOSSARY:

Intermediate results: Input data is transformed to output data through a series of intermediate steps; at each step data, or intermediate results, may be useful in determining where processing correctness begins and ends.

Display: print, hard copy, video, etc.

SPECIAL RESPONSE INSTRUCTIONS: If a module has no significant intermediate results (perhaps because of its inherent simplicity) which would be useful to know in order to test/retest the module, then answer A.

QUESTION DATA SHEET

Question Number S-82

QUESTION: Aids exist in or can be easily inserted into the module's source code for the purpose of tracing the logical flow of control.

CHARACTERISTIC: Instrumentation (control of instrumentation).

EXPLANATIONS: Language features such as conditional compilation can greatly influence the capability for a design to include good instrumentation aids for the purpose of collecting trace information or intermediate data results.

EXAMPLES: Event trace performance information is automatically collected and monitored by a separate program task.

GLOSSARY:

Aids: test probes, performance probes, etc. which may be in the form of language processor generated code, program diagnostic modules designed specifically for that purpose, or actual inline code which can be selectively activated by the user.

SPECIAL RESPONSE INSTRUCTIONS:

QUESTION DATA SHEET

Question Number S-83

QUESTION: Modularity as reflected in this module's source listing contributes to the maintainability of this module.

CHARACTERISTIC: General Questions.

EXPLANATIONS: Software possesses the characteristics of modularity to the extent a logical partitioning of software into parts, components, modules has occurred.

EXAMPLES: The software has been partitioned into easily comprehendable "sections". Each "section" is independent from every other "section" as much as is reasonable; i.e., to understand any given "section", requisite knowledge of other "sections" has been kept to a minimum.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general feeling about the modularity of the module source listing.

QUESTION DATA SHEET

Question Number S-84

QUESTION: Descriptiveness as reflected in this module's source listing contributes to the maintainability of this module.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of descriptiveness to the extent that is contains information regarding its objectives, assumptions, inputs, processing, outputs, components, revision status, etc.

EXAMPLES: Program objectives are explained, subprogram objectives are explained, communication links are specifically explained. Revision status of the documentation is clear, source listing revision status is clear.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general feeling about the descriptiveness of the module source listing.

QUESTION DATA SHEET

Question Number S-85

QUESTION: Consistency as reflected in this module's source listing and between the source listing and documentation contributes to the maintainability of this module.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of consistency to the extent the software products correlate and contain uniform notation, terminology and symbology.

EXAMPLES: Things are done similarly in different parts of the source listing. Once an individual learns how the source listings are organized, he can turn to any part of the listings and see exactly what he expects to see. A set of coding standards appears to have been set up and followed.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general feelings about the consistency of the module source listing.

QUESTION DATA SHEET

Question Number S-86

QUESTION: Simplicity as reflected in this module's source listing contributes to the maintainability of this module.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of simplicity to the extent that it lacks complexity in organization, language, and implementation techniques and reflects the use of singularity concepts and fundamental structures.

EXAMPLES: The organization of the source listings is logical. Uncomplicated, descriptive terminology is used throughout. Each section or part of the source code addresses a single subject and is minimally dependent upon other parts for a full understanding.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general impression about the simplicity of the module source listings.

QUESTION DATA SHEET

Question Number S-87

QUESTION: Expandability as reflected in this module's source listing contributes to the maintainability of this module.

CHARACTERISTIC: General questions.

EXPLANTIONS: Software possesses the characteristics of expandability to the extent that a physical change to information, computational functions, data storage or execution time can be easily accomplished.

EXAMPLES: Array sizes are paramaterized. Sub-units of code are modular such that they may be easily replaced or that new sub-units may be easily added. The code does not use all available computer memory.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general impression of the overall expandability of the module as reflected in the module source listing.

QUESTION DATA SHEET

Question Number S-88

QUESTION: Instrumentation as reflected in this module's source listing contributes to the maintainability of this module.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software possesses the characteristics of instrumentation to the extent it contains aids which enhance testing.

EXAMPLES: The listings contain comments about test cases. Specialized code exists which can be invoked for testing purposes. Comments exist which explain how to use DEBUG options. Comments exist which point out potential problems.

GLOSSARY:
DEBUG: Removal of BUGS.
BUG(s): Latent errors.

SPECIAL RESPONSE INSTRUCTIONS: Please give your feelings about the instrumentation of the software as reflected in the module source listings.

QUESTION DATA SHEET

Question Number S-89

QUESTION: Overall it appears that the characteristics of this module's source listing contribute to the maintainability of this module.

CHARACTERISTIC: General questions.

EXPLANATIONS: Software maintainability is defined as those characteristics of software which affect the ability of software engineers to:

(1) Correct errors.
(2) Add system capabilities through software changes.
(3) Delete features.
(4) Modify software to be compatible with hardware changes.

EXAMPLES: The module source listing is designed to aid you in maintenance of the subject software.

GLOSSARY:

SPECIAL RESPONSE INSTRUCTIONS: Please give your general impression as to how much the documentation would aid you in maintenance of the software under study.

PART III

SOFTWARE MAINTAINABILITY EVALUATOR'S HANDBOOK CROSS REFERENCES

A. GENERAL.

The following section contains information which should help the evaluator locate specific question subject matter and glossary definitions. The Question Index and Cross Reference contains question subjects listed alphabetically and cross-referenced to the question (and page number) addressing that question. The Glossary Index and Cross Reference lists all definitions in the handbook, arranged alphabetically and cross-referenced to the page containing that definition.

B. EVALUATOR'S HANDBOOK CROSS REFERENCES.

Question Index and Cross Reference

Glossary Index and Cross Reference